## Contents

1. **Contents** .......................... 3  
   1.1 What is PlatformIO? .................. 3  
   1.2 PlatformIO IDE ....................... 5  
   1.3 PlatformIO Core (CLI) ............... 7  
   1.4 PlatformIO Home ..................... 152  
   1.5 Tutorials and Examples ............. 164  
   1.6 “platformio.ini” (Project Configuration File) .................. 255  
   1.7 Environment variables .............. 293  
   1.8 Advanced Scripting ................ 297  
   1.9 Library Management .................. 311  
   1.10 Development Platforms .......... 328  
   1.11 Frameworks ......................... 531  
   1.12 Boards ................................ 717  
   1.13 Custom Platform & Board .......... 2409  
   1.14 Debugging ......................... 2414  
   1.15 Unit Testing ....................... 2583  
   1.16 Static Code Analysis .............. 2590  
   1.17 Remote Development ............... 2602  
   1.18 PlatformIO Account ............... 2607  
   1.19 Cloud & Desktop IDE ............. 2607  
   1.20 Continuous Integration .......... 2723  
   1.21 Compilation database **compile_commands.json** ................. 2747  
   1.22 Articles about us ................ 2747  
   1.23 Frequently Asked Questions ...... 2753  
   1.24 Release Notes ..................... 2761  
   1.25 Migrating from 4.x to 5.0 ........ 2764  

**Index** .................................. 2769
A place where Developers and Teams have true Freedom! No more vendor lock-in!

- Open source, maximum permissive Apache 2.0 license
- Cross-platform IDE and Unified Debugger
- Static Code Analyzer and Remote Unit Testing
- Multi-platform and Multi-architecture Build System
- Firmware File Explorer and Memory Inspection.

Social: LinkedIn | Twitter | Facebook | Community Forums
1.1 What is PlatformIO?

A place where Developers and Teams have true Freedom! No more vendor lock-in!

PlatformIO is a cross-platform, cross-architecture, multiple framework, professional tool for embedded systems engineers and for software developers who write applications for embedded products.

1.1.1 Awards

PlatformIO was nominated for the year’s best Software and Tools in the 2015/16 IoT Awards. A native PlatformIO IDE extension for Microsoft VSCode editor is the most rated/reviewed extension with over 800 five-star reviews in the whole Microsoft Marketplace. It also was installed by over 750,000 unique developers around the world.

1.1.2 Philosophy

PlatformIO’s unique philosophy in the embedded market provides developers with a modern integrated development environment (Cloud & Desktop IDE) that works cross-platform, supports many different software development kits...
(SDKs) or Frameworks, and includes sophisticated debugging (Debugging), unit testing (Unit Testing), automated code analysis (Static Code Analysis), and remote management (Remote Development). It is architected to maximize flexibility and choice by developers, who can use either graphical or command line editors (PlatformIO Core (CLI)), or both.

PlatformIO is a must-have tool for professional embedded systems engineers who develop solutions on more than one specific platform. In addition, by having a decentralized architecture, PlatformIO offers both new and existing developers a quick integration path for developing commercial-ready products, and reduces the overall time-to-market. And it runs on any one of your favorite modern operating systems (macOS, MS Windows, Linux, FreeBSD).

1.1.3 Technologies

PlatformIO applies the latest scalable and flexible software technology to the embedded market – an area traditionally served by complex software tools that experienced hardware engineers have learned over time (often painfully so). Instead, with PlatformIO, users can be hobbyists or professionals. They can import the classic Arduino “Blink” sketch or develop a sophisticated low-level embedded C program for a commercial product. Example code for any supported framework can be compiled and uploaded to a target platform in minutes.

The build system structure automatically tags software dependencies and applies them using a modular hierarchy that takes away the usual complexity and pain. Developers no longer have to manually find and assemble an environment of toolchains, compilers, and library dependencies to develop applications for a specific target. With PlatformIO, clicking the compile button will bring in all necessary dependencies automatically. It’s analogous to if you were a furniture designer, and your CAD program had a “build” button that caused a robot to fetch all the necessary pieces and fasteners and correctly assemble them.

PlatformIO Core (CLI) is a unique, developed-from-scratch build system that removes the usual pain of software integration, packaging, and library dependencies that developers encounter when they move beyond the bounds of a specific SDK or example embedded application. It can be used with a variety of code development environments and allows easy integration with numerous cloud platforms and web services feeds. The user experiences no barriers to getting started quickly: no license fees, no legal contracts. The user maintains full flexibility of the build environment because the tools are open source and permissively licensed (no permission needed to modify them, and no requirement to share changes.)

1.1.4 Problematic

- The main problem which repulses people from the embedded world is a complicated process to setup development software for a specific MCU/board: toolchains, proprietary vendor’s IDE (which sometimes isn’t free) and what is more, to get a computer with OS where that software is supported.
- Multiple hardware platforms (MCUs, boards) require different toolchains, IDEs, etc, and, respectively, spending time on learning new development environments.
- Finding proper libraries and code samples showing how to use popular sensors, actuators, etc.
- Sharing embedded projects between team members, regardless of an operating system they prefer to work with.

1.1.5 How does it work?

Without going too deep into PlatformIO implementation details, work cycle of the project developed using PlatformIO is as follows:

- Users choose board(s) interested in “platformio.ini” (Project Configuration File)
- Based on this list of boards, PlatformIO downloads required toolchains and installs them automatically.
• Users develop code and PlatformIO makes sure that it is compiled, prepared and uploaded to all the boards of interest.

1.2 PlatformIO IDE

PlatformIO IDE is the next-generation integrated development environment for IoT.

• Cross-platform build system without external dependencies to the OS software:
  – 800+ Boards
  – 35+ Development Platforms
  – 20+ Frameworks
• Debugging
• Remote Development
• Unit Testing
• C/C++ Intelligent Code Completion
• C/C++ Smart Code Linter for rapid professional development
• Library Manager for the hundreds popular libraries
• Multi-projects workflow with multiple panes
• Themes support with dark and light colors
• Serial Port Monitor
• Built-in Terminal with PlatformIO Core (CLI) and CLI tool (pio, platformio)
• Built-in PlatformIO Home.

We provide official packages (plugins, extensions) for the most popular IDEs and text editors.

Note: In our experience, VSCode offers better system performance, and users have found it easier to get started

1.2.1 PlatformIO for VSCode

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Python, PHP, Go) and runtimes (such as .NET and Unity).

Install PlatformIO for VSCode / Get started
1.2.2 PlatformIO for CLion

The **CLion** is a cross-platform C/C++ IDE for Linux, OS X, and Windows. CLion includes such features as a smart editor, code generation, code quality assurance, automated refactorings, on-the-fly code analysis, project manager, integrated version control systems and debugger.

*Install PlatformIO for CLion / Get started*
1.3 PlatformIO Core (CLI)

PlatformIO Core (CLI tool) is a heart of whole PlatformIO ecosystem and consists of

- Multi-platform Build System
- Development platform and package managers
- Library Management
- Library Dependency Finder (LDF)
- Serial Port Monitor
- Integration components (Cloud & Desktop IDE and Continuous Integration).

PlatformIO Core is written in Python and works on Windows, macOS, Linux, FreeBSD and ARM-based credit-card sized computers (Raspberry Pi, BeagleBone, CubieBoard, Samsung ARTIK, etc.).

PlatformIO Core provides a rich and documented Command Line Interface (CLI). The other PlatformIO-based software and IDEs are based on PlatformIO Core CLI, such as PlatformIO IDE. In other words, they wrap PlatformIO Core with own GUI.

Note: Please note that you do not need to install PlatformIO Core if you are going to use PlatformIO IDE. PlatformIO Core is built into PlatformIO IDE and you will be able to use it within PlatformIO IDE Terminal.

If you need PlatformIO Core commands outside PlatformIO IDE, please Install Shell Commands.

### 1.3.1 Demo

**Contents**

- **“Blink Project”**
  - Used in demo
- **Platform Manager**
  - Used in demo
- **Library Manager**
  - Used in demo
- **Over-the-Air update for ESP8266**
  - Used in demo

**“Blink Project”**

**Used in demo**

1. Source code of Wiring Blink Example
2. `pio run` command
3. `pio run -t upload` command.

**Platform Manager**
Used in demo

1. *Platform Manager CLI*
2. `$pio platform list` command
3. `$pio platform search avr` command
4. `$pio platform show teensy` command
5. `$pio platform update` command.

Library Manager

Used in demo

1. *Library Manager CLI*
2. `$pio lib search 1-wire` command
3. `$pio lib install 54` command
4. `$pio lib search -f mbed` command
5. `$pio lib search -k rf` command
6. `$pio lib search radiohead` command
7. `$pio lib install 124 –version “1.40”` command
8. `$pio lib show 124` command
9. `$pio lib update` command.
Over-the-Air update for ESP8266

1. `pio run` command
2. `pio run -t upload` command.

### 1.3.2 Installation

**Note:** Please note that you do not need to install *PlatformIO Core (CLI)* if you are going to use *PlatformIO IDE*. *PlatformIO Core (CLI)* is built into PlatformIO IDE and you will be able to use it within PlatformIO IDE Terminal.

If you need *PlatformIO Core (CLI)* outside PlatformIO IDE, please *Install Shell Commands*.

*PlatformIO Core* is written in *Python* and works on *Windows*, *macOS*, *Linux*, *FreeBSD* and *ARM*-based credit-card sized computers (*Raspberry Pi*, *BeagleBone*, *CubieBoard*, *Samsung ARTIK*, etc.).

- **System requirements**
- **Installation Methods**
  - Installer Script
    - Super-Quick (*Mac / Linux*)
    - Local Download (*Mac / Linux / Windows*)
  - Python Package Manager
System requirements

**Operating System**  Windows, macOS, Linux, FreeBSD, Linux ARMv6+

**Python Interpreter**  Python 3.6+ or above. See detailed instruction on how to *Install Python Interpreter*.

**Terminal Application**  All commands below should be executed in Command-line application (Terminal). For macOS and Linux OS - *Terminal* application, for Windows OS – *cmd.exe* application.

**Access to Serial Ports (USB/UART)**  *Windows Users:*  Please check that you have correctly installed USB driver from board manufacturer

*Linux Users:*

- Please install *99-platformio-udev.rules*
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

Installation Methods

Please *choose ONE of* the following methods:

- **Installer Script**
  - *Super-Quick (Mac / Linux)*
  - *Local Download (Mac / Linux / Windows)*

- **Python Package Manager**
macOS Homebrew

Installer Script

**Warning:** PlatformIO **DOES NOT** require administrative/sudo permissions. Please install using default user account **WITHOUT EXTRA PERMISSIONS.**

**Super-Quick (Mac / Linux)**

To install or upgrade **PlatformIO Core** paste that at a *Terminal* prompt:

```bash
python3 -c "$(curl -fsSL https://raw.githubusercontent.com/platformio/platformio/master/scripts/get-platformio.py)"
```

# or using `curl`

```
python3 get-platformio.py
```

# or using `wget`

```
python3 get-platformio.py
```

**Local Download (Mac / Linux / Windows)**

To install or upgrade **PlatformIO Core**, download (save as...) *get-platformio.py* script. Then run the following:

```bash
# change directory to folder where is located downloaded "get-platformio.py"
cd /path/to/dir/where/is/located/get-platformio.py/script

# run it
python get-platformio.py
```

On **Windows OS** it may look like:

```bash
# change directory to folder where is located downloaded "get-platformio.py"
cd C:/path/to/dir/where/is/located/script/get-platformio.py

# run it
python.exe get-platformio.py
```

**Note:** If you need to have access to **platformio** or **platformio.exe** commands from other applications or terminal in your OS, please *Install Shell Commands.*
Python Package Manager

**Warning:** We recommend using this method ONLY FOR Continuous Integration use cases or where you have full permissions to install PlatformIO Core into the global scope of your OS.

For personal using, and avoiding maintenance and upgrade issues, we HIGHLY RECOMMEND using Installer Script which installs PlatformIO Core into an isolated virtual environment and does not affect your OS.

The latest stable version of PlatformIO Core may be installed or upgraded via Python Package Manager (pip) as follows:

```
pip install -U platformio
```

macOS Homebrew

The latest stable version of PlatformIO may be installed or upgraded via macOS Homebrew Packages Manager (brew) as follows:

```
brew install platformio
```

Development Version

**Warning:** If you use PlatformIO IDE, please enable development version:

- **PlatformIO IDE for Atom:** “Menu PlatformIO: Settings > PlatformIO IDE > Use development version of PlatformIO Core”
- **VSCode:** Set `platformio-ide.useDevelopmentPIOCore` to true in Settings.

Install the latest PlatformIO from the develop branch:

```
# uninstall existing version
pip uninstall platformio

# install the latest development version of PlatformIO
pip install -U https://github.com/platformio/platformio-core/archive/develop.zip
```

If you want to be up-to-date with the latest develop version of PlatformIO, then you need to re-install PlatformIO each time you see a new commits in PlatformIO GitHub repository (branch: develop) like so:

```
pip install -U https://github.com/platformio/platformio-core/archive/develop.zip
```

Or:

```
PIO upgrade --dev
```

To revert to the latest stable version:

```
pip uninstall platformio
pip install -U platformio
```
Install Shell Commands

PlatformIO Core (CLI) consists of 2 standalone tools in a system:

- `platformio` or `pio` (short alias) - CLI Guide
- `piodebuggdb` - alias of `pio debug`

If you have PlatformIO IDE already installed, you do not need to install PlatformIO Core (CLI) separately. Just link these tools with your shell:

- **Unix and Unix-like**
  - **Method 1**
  - **Method 2**
- **Windows**

### Unix and Unix-like

In Unix and Unix-like systems, there are multiple ways to achieve this.

#### Method 1

You can export PlatformIO executables’ directory to the PATH environmental variable. This method will allow you to execute `platformio` commands from any terminal emulator as long as you’re logged in as the user PlatformIO is installed and configured for.

If you use Bash as your default shell, you can do it by editing either `~/.profile` or `~/.bash_profile` and adding the following line:

```
export PATH=$PATH:~/.platformio/penv/bin
```

If you use Zsh, you can either edit `~/.zprofile` and add the code above, or for supporting both, Bash and Zsh, you can first edit `~/.profile` and add the code above, then edit `~/.zprofile` and add the following line:

```
emulate sh -c '. ~/.profile'
```

After everything’s done, just restart your session (log out and log back in) and you’re good to go.

If you don’t know the difference between the two, check out this page.

#### Method 2

You can create system-wide symlinks. This method is not recommended if you have multiple users on your computer because the symlinks will be broken for other users and they will get errors while executing PlatformIO commands. If that’s not a problem, open your system terminal app and paste these commands (MAY require administrator access `sudo`):

```
ln -s ~/.platformio/penv/bin/platformio /usr/local/bin/platformio
ln -s ~/.platformio/penv/bin/pio /usr/local/bin/pio
ln -s ~/.platformio/penv/bin/piodebuggdb /usr/local/bin/piodebuggdb
```
After that, you should be able to run PlatformIO from terminal. No restart is required.

**Windows**

Please read one of these instructions [How do I set or change the PATH system variable?](#).

You need to edit system environment variable called `Path` and append

```bash
C:\Users\UserName\.platformio\penv\Scripts; path in the beginning of a list (please replace `UserName` with your account name).
```

**Uninstall PlatformIO Core and dependent packages**

- Uninstall PlatformIO Core tool

```bash
# uninstall standalone PlatformIO Core installed via `pip`
pip uninstall platformio

# uninstall Homebrew's PlatformIO Core (only macOS users if you installed it via Homebrew before)
brew uninstall platformio
```

- Dependent packages, global libraries are installed to `core_dir` folder (in user’s HOME directory). Just remove it.

**Integration with custom applications (extensions, plugins)**

We recommend using PlatformIO Core Installer Script when you integrate PlatformIO Core into an application, such as extension or plugin for IDE. Examples that use this installer are:

- `platformio-node-helpers`, is used by PlatformIO IDE for VSCode and PlatformIO IDE for Atom

**Prerequisite**

**Python Interpreter**

PlatformIO Core Installer Script is written in Python and is compatible with Python 2.7+ and Python 3.5+. We highly recommend using the latest Python 3.

Python is installed by default on the most popular Unix OS (macOS, Linux, FreeBSD). If there is no Python on a user machine (you can check running `python --version`), we have 2 options:

1. Ask the user to install Python 3 using our guide [Install Python Interpreter](#).
2. You can automatically Download Portable Python 3 and unpack it in a cache folder of your application. Later, you can use `unpacked_portable_python_dir/python.exe` for the installer script.

**Installer Script**

There are 2 options on how to work with PlatformIO Core Installer Script:

1. Bundle `get-platformio.py` file into your application
2. Download `get-platformio.py` file on demand.
In both cases, you will need to have `get-platformio.py` script on the end-user machine. You can copy or download it to a cache/temporary folder.

A list of arguments and options for the installer script is available via

```
python get-platformio.py --help
```

**Workflow**

We will describe a simple workflow on how to automatically install PlatformIO Core (CLI) for end-user of your application/extension. We assume that `get-platformio.py` script is already copied/downloaded and exists on the end-user machine. See above how to get it.

**Step 1. Where is PlatformIO Core installed?**

You should check the PlatformIO Core installation state **each time** when the user starts your application. You need to call the Installer Script with `check core` arguments:

```
python get-platformio.py check core
```

This command returns 0 “exit code” when PlatformIO Core is already installed and is ready for use, otherwise, the non-zero code of subprocess will be returned and you need to install PlatformIO Core (see **Step #2** below).

If you need to have full information about PlatformIO Core installation state, please run with `--dump-state` option and specify a folder or a full path where to save data in JSON format:

```
get-platformio.py check core --dump-state tmpdir/pioinstaller-state.json
```

Now, read JSON file and use `platformio_exe` binary to call PlatformIO Core using CLI (see CLI Guide). You can also export `penv_bin_dir` into system environment PATH variable and `platformio` command will be available without a full path.

Example of `pioinstaller-state.json` run on macOS:

```
{
    "cache_dir": "/Users/Freedom/.platformio/.cache",
    "core_dir": "/Users/Freedom/.platformio",
    "core_version": "4.3.1",
    "installer_version": "0.2.0",
    "is_develop_core": false,
    "penv_bin_dir": "/Users/Freedom/.platformio/penv/bin",
    "penv_dir": "/Users/Freedom/.platformio/penv",
    "platformio_exe": "/Users/Freedom/.platformio/penv/bin/platformio",
    "python_exe": "/Users/Freedom/.platformio/penv/bin/python",
    "system": "darwin_x86_64"
}
```

**Step 2. Install PlatformIO Core**

To install PlatformIO Core into the virtual environment in an automatic mode, please call installer script without any arguments:

```
python get-platformio.py
```
Available options:

- `--verbose`, verbose output
- `--dev`, install the latest development version of PlatformIO Core
- `--ignore-python`, a path to Python to be ignored (multiple options and unix wildcards are allowed)

More options are available at `python get-platformio.py --help`

Installer Script will return exit code 0 on success, otherwise non-zero code and error explanation.

Next time just use again `python get-platformio.py check core` as described in Step #1 (see above).

**Troubleshooting**

**Note:** Linux OS: Don’t forget to install “udev” rules file `99-platformio-udev.rules` (an instruction is located in the file).

**Windows OS:** Please check that you have correctly installed USB driver from board manufacturer

For further details, frequently questions, known issues, please refer to *Frequently Asked Questions*.

If you find any issues with PlatformIO Core Installer Script, please report to https://github.com/platformio/platformio-core-installer/issues

### 1.3.3 Quick Start

This tutorial introduces you to the basics of **PlatformIO Core (CLI)** Command Line Interface (CLI) workflow and shows you a creation process of a simple cross-platform “Blink” Project. After finishing you will have a general understanding of how to work with the multiple development platforms and embedded boards.

**Setting Up the Project**

*PlatformIO Core (CLI)* provides special `pio project init` command for configuring your projects. It allows one to initialize new empty project or update existing with the new data.

What is more, `pio project init` can be used for **Cloud & Desktop IDE**. It means that you will be able to import pre-generated PlatformIO project using favorite IDE and extend it with the professional instruments for IoT development.

This tutorial is based on the next popular embedded boards and development platforms using *Arduino*:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Board</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel AVR</td>
<td>Arduino Uno</td>
<td>Arduino</td>
</tr>
<tr>
<td>ESPressif 8266</td>
<td>NodeMCU 1.0 (ESP-12E Module)</td>
<td>Arduino</td>
</tr>
<tr>
<td>Teensy</td>
<td>Teensy 3.1 / 3.2</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

**Board Identifier**

`pio project init` command requires to specify board identifier ID. It can be found using *Boards* catalog, *Boards Explorer* or `pio boards` command. For example, using `pio boards` let’s try to find Teensy boards:
According to the table above the ID for Teensy 3.1 / 3.2 is teensy31. Also, the ID for Arduino Uno is uno and for NodeMCU 1.0 (ESP-12E Module) is nodemcuv2.

Initialize Project

PlatformIO ecosystem contains big database with pre-configured settings for the most popular embedded boards. It helps you to forget about installing toolchains, writing build scripts or configuring uploading process. Just tell PlatformIO the Board ID and you will receive full working project with pre-installed instruments for the professional development.

1. Create empty folder where you are going to initialize new PlatformIO project. Then open system Terminal and change directory to it:

```
# create new directory
> mkdir path_to_the_new_directory

# go to it
> cd path_to_the_new_directory
```

2. Initialize project for the boards mentioned above (you can specify more than one board at time):

```
> pio project init --board uno --board nodemcuv2 --board teensy31
```

The current working directory *** will be used for the new project. You can specify another project directory via `pio project init -d %PATH_TO_THE_PROJECT_DIR%` command.

The next files/directories will be created in *** platformio.ini - Project Configuration File. |-> PLEASE EDIT ME <-| src - Put your source files here lib - Put here project specific (private) libraries Do you want to continue? [y/N]: y Project has been successfully initialized! Useful commands:

```
> pio run - process/build project from the current directory
> pio run --target upload or [pio run -t upload] - upload firmware to embedded board
> pio run --target clean - clean project (remove compiled files)
```

Congrats! You have just created the first PlatformIO based Project with the next structure:

- "platformio.ini" (Project Configuration File)
- src directory where you should place source code (*.h, *.c, *.cpp, *.S, *.ino, etc.)
• `lib` directory can be used for the project specific (private) libraries. More details are located in `lib/README` file.

• Miscellaneous files for VCS and Continuous Integration support.

Note: If you need to add new board to the existing project please use `pio project init` again.

The result of just generated `platformio.ini`:

```
; PlatformIO Project Configuration File
;
; Build options: build flags, source filter, extra scripting
; Upload options: custom port, speed and extra flags
; Library options: dependencies, extra library storages
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:uno]
platform = atmelavr
framework = arduino
board = uno

[env:nodemcuv2]
platform = espressif8266
framework = arduino
board = nodemcuv2

[env:teensy31]
platform = teensy
framework = arduino
board = teensy31
```

Now, we need to create `main.cpp` file and place it to `src` folder of our newly created project. The contents of `src/main.cpp`:

```
/**
 * Blink
 *
 * Turns on an LED on for one second,
 * then off for one second, repeatedly.
 */
#include "Arduino.h"

#ifndef LED_BUILTIN
#define LED_BUILTIN 13
#endif

void setup()
{
    // initialize LED digital pin as an output.
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
    // turn the LED on (HIGH is the voltage level) (continues on next page)
```
The final Project structure:

```
project_dir
  lib
    README
    platformio.ini
  src
    main.cpp
```

### Process Project

*PlatformIO Core (CLI)* provides special *pio run* command to process project. If you call it without any arguments, PlatformIO Build System will process all project environments (which were created per each board specified above). Here are a few useful commands:

- `pio run`. Process (build) all environments specified in “platformio.ini” (Project Configuration File)
- `pio run --target upload`. Build project and upload firmware to the all devices specified in “platformio.ini” (Project Configuration File)
- `pio run --target clean`. Clean project (delete compiled objects)
- `pio run -e uno`. Process only uno environment
- `pio run -e uno -t upload`. Build project only for uno and upload firmware.

Please follow to *pio run --list-targets* documentation for the other targets.

Finally, demo which demonstrates building project and uploading firmware to Arduino Uno:

**Further Reading**

- Project examples
- *CLI Guide* for *PlatformIO Core (CLI)* commands

### 1.3.4 CLI Guide

**Contents**
Usage

pio [OPTIONS] COMMAND
# alias of "pio"
platformio [OPTIONS] COMMAND

Options

--no-ansi
Do not print ANSI control characters.

See also PLATFORMIO_NO_ANSI and PLATFORMIO_FORCE_ANSI environment variables.

--version
Show the version of PlatformIO

--help, -h
Show help for the available options and commands

$ pio --help
$ pio COMMAND --help

Commands

pio access

New in version 5.0.
Set access level on published resources (packages) in the registry.

You must have privileges to set the access of a resource:

- You are an owner of published resources
- You are a member of the team that owns a resource
- You have been given “maintainer” privileges for a package, either as a member of a team or directly as an owner.

Management of teams and team memberships is done with the pio team command.
To print all available commands and options use:

pio access --help
pio access COMMAND --help
pio access grant

New in version 5.0.

Usage

```
pio access grant [OPTIONS] LEVEL <ORGNAME:TEAMNAME>|<USERNAME> URN
```

Description

Add the ability of users and teams to have “LEVEL” access to a resource.

The next levels are supported:

- **admin** Administrative privileges. Can perform all available operations on the particular resource.
- **maintainer** Read-write access. Can update a resource, publish a new version, unpublish existing version.
- **guest** Read-only access. It’s automatically granted for the public resources.

You can grant access to the team members using the next format `<orgname:teamname>`. See the examples below.

Options

- **--urn-type**

  Resource type in URN form. Default is set to `prn:reg:pkg` which means to grant an access to the package in the PlatormIO Registry.

Examples

1. Grant “maintainer” access to a resource for the “bob” user:

   ```
   > pio access grant maintainer bob prn:reg:pkg:8036:platform
   Access for resource "prn:reg:pkg:8036:platform" has been granted for "bob"
   ```

2. Add the ability of PlatformIO’s “developer” team to have “admin” access to a resource.
Access for resource "prn:reg:pkg:8036:platform" has been granted for "platformio:developers"

See Also

- pio access revoke

pio access list

New in version 5.0.

Contents

- pio access list
  - Usage
  - Description
    * Options
  - Examples

Usage

```
pio access list [OPTIONS] [OWNER]
```

Description

Show all of the resources (packages) a user or a team can access, along with the access level, except for read-only public resources (it won’t print the whole registry listing).

To list resources by the specified owner, type owner’s “username” as an argument to this command.

See the examples below.

Options

--urn-type

Resource type in URN form. Default is set to prn:reg:pkg which means to list the packages from the registry.

--json-output

Return the output in JSON format
Examples

1. List all accessible resources:

```plaintext
> pio access list
...
atmelsam
--------
URN: prn:reg:pkg:8007:platform
Owner: platformio
Access level(s): Admin
espressif8266
-------------
URN: prn:reg:pkg:8008:platform
Owner: platformio
Access level(s): Admin
chipsalliance
----------
URN: prn:reg:pkg:8036:platform
Owner: platformio
Access level(s): Admin
contrib-piohome
---------------
Owner: platformio
Access level(s): Admin
contrib-pysite
------------
URN: prn:reg:pkg:8038:tool
Owner: platformio
Access level(s): Admin
...
```

2. List all accessible resources by specific owner:

```plaintext
> pio access list platformio
...
tool-scons
--------
URN: prn:reg:pkg:8192:tool
Owner: platformio
Access level(s): Admin
tool-simavr
---------
URN: prn:reg:pkg:8193:tool
Owner: platformio
Access level(s): Admin
```

(continues on next page)
pio access private

New in version 5.0.

Usage

```bash
pio access private [OPTIONS] URN
```

Description

Set a resource to be either privately accessible (restricted access to particular users or team members).

Options

`--urn-type`

Resource type in URN form. Default is set to `prn:reg:pkg` which means to grant access to the package in the registry.

Examples

```bash
> pio access private prn:reg:pkg:8036:platform
The resource "prn:reg:pkg:8036:platform" has been successfully updated.
```

See Also

- `pio access public`

1.3. PlatformIO Core (CLI)
pio access public

New in version 5.0.

Contents

• pio access public
  – Usage
  – Description
  • Options
  – Examples
  – See Also

Usage

pio access public [OPTIONS] URN

Description

Set a resource to be either publicly accessible.

Options

--urn-type

Resource type in URN form. Default is set to prn:reg:pkg which means to grant access to the package in the registry.

Examples

> pio access public prn:reg:pkg:8036:platform
The resource "prn:reg:pkg:8036:platform" has been successfully updated.

See Also

• pio access private

pio access revoke

New in version 5.0.
Usage

```bash
pio access revoke [OPTIONS] <ORGNAME:TEAMNAME>|<USERNAME> URN
```

Description

Remove the ability of users and teams to have access to a resource.
You can revoke access for the team members using the next format `orgname:teamname`.
See the examples below.

Options

```
--urn-type
```
Resource type in URN form. Default is set to `prn:reg:pkg` which means to grant an access to the package in the registry.

Examples

1. Revoke access from a resource for the “bob” user:

```bash
> pio access revoke bob prn:reg:pkg:8036:platform
Access for resource "prn:reg:pkg:8036:platform" has been revoked for "bob"
```

2. Add the ability of PlatformIO’s “developer” team to have “admin” access to a resource.

```bash
> pio access revoke platformio:developers prn:reg:pkg:8036:platform
Access for resource "prn:reg:pkg:8036:platform" has been revoked for
→"platformio:developers"
```

See Also

```
• pio access grant
```
platformio documentation, release 5.0.5a1

pio account

cli helper command for platformio account.
to print all available commands and options use:

```
pio account --help
pio account COMMAND --help
```

pio account destroy

contents

• pio account destroy
  – usage
  – description

usage

```
pio account destroy [OPTIONS]
```

description

permanently remove account and related data (organizations, teams, resources).

WARNING! it can not be restored.

pio account forgot

contents

• pio account forgot
  – usage
  – description
    * options

usage

```
pio account forgot [OPTIONS]
```
Description

Allows you to reset password for PlatformIO Account using username or email which were specified for registration.

Options

--username, -u

Username or email. You can omit this option and enter username or email in Forgot Wizard later.

pio account login

Usage

pio account login [OPTIONS]

Description

Log in to PlatformIO Account. If you are not able to provide authentication credentials manually you can use PLATFORMIO_AUTH_TOKEN. This is very useful for Continuous Integration systems and Remote Development operations.

Options

--username, -u

Username or email. You can omit this option and enter username or email in Login Wizard later.

--password, -p

You can omit this option and enter securely password in Login Wizard later.

pio account logout

Contents

• pio account logout
Usage

pio account logout

Description

Log out of PlatformIO Account.

pio account password

Contents

• pio account password
  – Usage
  – Description

Usage

pio account password

Description

Change password for PlatformIO Account.

pio account register

Contents

• pio account register
  – Usage
  – Description
  • Options
Usage

```bash
pio account register [OPTIONS]
```

Description

Create a new PlatformIO Account.

Options

You can omit these options and enter them later in Register Wizard.

```bash
--username, -u
```
A username. You can use it later for `pio account login`, `pio account update`, and `pio account forgot` commands.
The username must contain at least 4 characters including single hyphens, and cannot begin or end with a hyphen.

```bash
--email, -e
```
An email. Please enter existing email, you will receive a confirmation letter.

```bash
--password, -p
```
A password. You will need it for `pio account login`, `pio account password`, `pio account token`, and `pio account update` commands.

```bash
--firstname
```
A first name.

```bash
--lastname
```
A last name.

**pio account show**
**Description**

Show detailed information about *PlatformIO Account*:

- Active subscriptions
- Available packages and services

**Options**

`--json-output`

Return the output in JSON format

**pio account token**

**Usage**

`pio account token`

**Description**

Get or regenerate Personal Authentication Token. It is very useful for *Continuous Integration* systems, *Remote Development* operations where you are not able to authorize manually.

PlatformIO handles Personal Authentication Token from environment variable `PLATFORMIO_AUTH_TOKEN`.

**Options**

`--regenerate`

If this option is specified a new authentication token will be generated.

`--json-output`

Return the output in JSON format
pio account update

Usage
pio account update [OPTIONS]

Description
Update PlatformIO Account profile.

Options
You can omit these options and enter them later in update Wizard.

--username, -u
A username that must contain at least 4 characters including single hyphens, and cannot begin or end with a hyphen.

--email, -e
An email. Please enter existing email, you will receive a confirmation letter.

--firstname
A first name.

--lastname
A last name.

--current-password
A current password to confirm this operation.

pio boards

Contents

• pio boards
  – Usage
  – Description
Usage

`pio boards [OPTIONS] [FILTER]`

Description

List pre-configured Embedded Boards

Options

`--installed`
List boards only from the installed platforms

`--json-output`
Return the output in JSON format

Examples

1. Show all available pre-configured embedded boards

   ```
   > pio boards
   Platform: atmelavr
   # ID     MCU       Frequency  Flash  RAM  Name
   #----------------------------------------------------------
   # btatmega168 atmega168 16MHz  14K  1K  Arduino BT ATmega168
   # btatmega328 atmega328p 16MHz  28K  2K  Arduino BT ATmega328
   # diecimilaatmega168 atmega168 16MHz  14K  1K  Arduino Duemilanove or
   #  → Diecimila ATmega168
   # diecimilaatmega328 atmega328p 16MHz  30K  2K  Arduino Duemilanove or
   #  → Diecimila ATmega328
   # esplora atmega32u4 16MHz  28K  2K  Arduino Esplora
   # ethernet atmega328p 16MHz  31K  2K  Arduino Ethernet
   # ...
   ```

2. Filter Arduino-based boards

   ```
   > pio boards arduino
   Platform: atmelavr
   # ID     MCU       Frequency  Flash  RAM  Name
   #----------------------------------------------------------
   # btatmega168 atmega168 16MHz  14K  1K  Arduino BT ATmega168
   # btatmega328 atmega328p 16MHz  28K  2K  Arduino BT ATmega328
   ```

(continues on next page)
3. Filter mbed-enabled boards

```bash
> pio boards mbed
```

Platform: freescalekinetis

<table>
<thead>
<tr>
<th>ID</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>frdm_k20d50m</td>
<td>mk20dx128v1h5</td>
<td>48MHz</td>
<td>128K</td>
<td>16K</td>
<td>Freescale Kinetis FRDM-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>K20D50M</td>
</tr>
<tr>
<td>frdm_k22f</td>
<td>mk22fn512v1h12</td>
<td>120MHz</td>
<td>512K</td>
<td>128K</td>
<td>Freescale Kinetis FRDM-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>K22F</td>
</tr>
</tbody>
</table>

Platform: nordicnrf51

<table>
<thead>
<tr>
<th>ID</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallBotBLE</td>
<td>nrf51822</td>
<td>16MHz</td>
<td>128K</td>
<td>16K</td>
<td>JKSoft Wallbot BLE</td>
</tr>
<tr>
<td>nrf51_dk</td>
<td>nrf51822</td>
<td>32MHz</td>
<td>256K</td>
<td>32K</td>
<td>Nordic nRF51-DK</td>
</tr>
</tbody>
</table>

Platform: nxplpc

<table>
<thead>
<tr>
<th>ID</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>blueboard_lpc11u24</td>
<td>lpc11u24</td>
<td>48MHz</td>
<td>32K</td>
<td>8K</td>
<td>BlueBoard-LPC11U24</td>
</tr>
<tr>
<td>dipcortexm0</td>
<td>lpc11u24</td>
<td>50MHz</td>
<td>32K</td>
<td>8K</td>
<td>DipCortex M0</td>
</tr>
<tr>
<td>lpc11u35</td>
<td>lpc11u35</td>
<td>48MHz</td>
<td>64K</td>
<td>10K</td>
<td>EA LPC11U35 QuickStart,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Platform: ststm32

<table>
<thead>
<tr>
<th>ID</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>disco_f401vc</td>
<td>stm32f401vct6</td>
<td>84MHz</td>
<td>256K</td>
<td>64K</td>
<td>32F401CDISCOVERY</td>
</tr>
<tr>
<td>nucleo_f030r8</td>
<td>stm32f030r8t6</td>
<td>48MHz</td>
<td>64K</td>
<td>8K</td>
<td>ST Nucleo F030R8</td>
</tr>
</tbody>
</table>

4. Filter boards which are based on ATmega168 MCU

```bash
> pio boards atmega168
```

Platform: atmelavr

<table>
<thead>
<tr>
<th>ID</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>btatmega168</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino BT ATmega168</td>
</tr>
</tbody>
</table>

(continues on next page)
(continued from previous page)

<table>
<thead>
<tr>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diecimila</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Duemilanove or</td>
</tr>
<tr>
<td>mini</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Mini ATmega168</td>
</tr>
<tr>
<td>atmega168</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino NG or older</td>
</tr>
<tr>
<td>nano</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Nano ATmega168</td>
</tr>
<tr>
<td>pro8MHz</td>
<td>atmega168</td>
<td>8MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Pro or Pro Mini</td>
</tr>
<tr>
<td>pro16MHz</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Pro or Pro Mini</td>
</tr>
<tr>
<td>lilypad</td>
<td>atmega168</td>
<td>8MHz</td>
<td>14K</td>
<td>1K</td>
<td>LilyPad Arduino ATmega168</td>
</tr>
<tr>
<td>168pa16m</td>
<td>atmega168p</td>
<td>16MHz</td>
<td>15K</td>
<td>1K</td>
<td>Microduino Core</td>
</tr>
<tr>
<td>168pa8m</td>
<td>atmega168p</td>
<td>8MHz</td>
<td>15K</td>
<td>1K</td>
<td>Microduino Core</td>
</tr>
</tbody>
</table>

5. Show boards by **TI MSP430**

```bash
> pio boards timsp430
```

Platform: timsp430

<table>
<thead>
<tr>
<th>ID</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>lmsp430fr5739</td>
<td>msp430fr5739</td>
<td>16MHz</td>
<td>15K</td>
<td>1K</td>
<td>FraunchPad w/ msp430fr5739</td>
</tr>
<tr>
<td>lmsp430f5529</td>
<td>msp430f5529</td>
<td>16MHz</td>
<td>128K</td>
<td>1K</td>
<td>LaunchPad w/ msp430f5529</td>
</tr>
<tr>
<td>lmsp430f5529_25</td>
<td>msp430f5529_25</td>
<td>25MHz</td>
<td>128K</td>
<td>1K</td>
<td>LaunchPad w/ msp430f5529</td>
</tr>
<tr>
<td>lmsp430fr5969</td>
<td>msp430fr5969</td>
<td>8MHz</td>
<td>64K</td>
<td>1K</td>
<td>LaunchPad w/ msp430fr5969</td>
</tr>
<tr>
<td>lmsp430g2231</td>
<td>msp430g2231</td>
<td>1MHz</td>
<td>2K</td>
<td>128B</td>
<td>LaunchPad w/ msp430g2231</td>
</tr>
<tr>
<td>lmsp430g2452</td>
<td>msp430g2452</td>
<td>16MHz</td>
<td>8K</td>
<td>256B</td>
<td>LaunchPad w/ msp430g2452</td>
</tr>
<tr>
<td>lmsp430g2553</td>
<td>msp430g2553</td>
<td>16MHz</td>
<td>16K</td>
<td>512B</td>
<td>LaunchPad w/ msp430g2553</td>
</tr>
</tbody>
</table>

**pio check**

Helper command for **Static Code Analysis.**

**Contents**

- **pio check**
  - Usage
  - Description
  - Options
  - Examples
Usage

```
pio check [OPTIONS]
```

Description

Perform static analysis check on PlatformIO based project. By default Cppcheck analysis tool is used.

Options

`-e, --environment`
Process specified environments.

`--pattern`
You can specify which source files or folders should be included/excluded from check process. By default only `src_dir` and `include_dir` are checked. Multiple `--pattern` options and GLOB Patterns are allowed.
Example: `pio check --pattern="tests" --pattern="src/*.cpp"

`--flags`
Specify additional flags that need to be passed to the analysis tool. If multiple tools set in `check_tool` option, the flags are passed to all of them. Individual flags for each tool can be added using a special suffix with the tool name.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>--addon=&lt;addon&gt;</td>
<td>Execute addon. i.e. cert.</td>
</tr>
<tr>
<td>-D&lt;ID&gt;</td>
<td>Define preprocessor symbol.</td>
</tr>
</tbody>
</table>

Multiple `--flags` options are allowed.
Example: `pio check --flags "-DDEBUG cppcheck: --std=c++11 --platform=avr8"

`--severity`
Specify the Defect severity types which will be reported by the Check tools. Possible values described in Defect severity section. Multiple `--severity` options are allowed.
Example: `pio check --severity=high`

`-d, --project-dir`
Specify the path to project directory. By default, `--project-dir` is equal to the current working directory (CWD).

`-c, --project-conf`
Process project with a custom “platformio.ini” (Project Configuration File).

`--json-output`
Return the output in JSON format.

`--fail-on-defect

1.3. PlatformIO Core (CLI)
Fail (exit with non-zero code) if there is a defect found with specified severity. By default exit code is the same as the exit code returned by a tool selected for performing check. Possible values described in *Defect severity* section. Multiple `--fail-on-defect` options are allowed.

Example: `pio check --fail-on-defect=low --fail-on-defect=medium`

`-s, --silent`
Suppress progress reporting and show only defects with high severity. See *Defect severity*.

`-v, --verbose`
Show detailed information when processing environments.

This option can also be set globally using `force_verbose` setting or by environment variable `PLATFORMIO_SETTING_FORCE_VERBOSE`.

**Examples**

For the examples please follow to *Static Code Analysis* page.

`pio ci`

**Contents**

- `pio ci`
  - *Usage*
  - *Description*
  - *Options*
  - *Examples*

**Usage**

`pio ci [OPTIONS] [SRC]`

**Description**

`pio ci` command is conceived of as “hot key” for building project with arbitrary source code structure. In a nutshell, using SRC and `pio ci --lib` contents PlatformIO initializes via `pio project init` new project in `pio ci --build-dir` with the build environments (using `pio ci --board` or `pio ci --project-conf`) and processes them via `pio run` command.

`pio ci` command accepts multiple SRC arguments, `pio ci --lib` and `pio ci --exclude` options which can be a path to directory, file or *Glob Pattern*. Also, you can omit SRC argument and set path (multiple paths are allowed denoting with :) to `PLATFORMIO_CI_SRC` Environment variable.

For more details as for integration with the popular Continuous Integration Systems please follow to *Continuous Integration* page.
Note: `pio ci` command is useful for library developers. It allows one to build different examples without creating own project per them. Also, it is possible to upload firmware to the target device. In this case, you need to pass additional option `--project-option="targets=upload"`. What is more, you can specify custom upload port using `--project-option="upload_port=<port>"` option. See `pio ci --project-option` for details.

### Options

-b, --board
Build project with automatically pre-generated environments based on board settings.
For more details please look into `pio project init --board`.

--build-dir
Path to directory where PlatformIO will initialise new project. By default it’s temporary directory within your operating system.

Note: This directory will be removed at the end of build process. If you want to keep it, please use `pio ci --keep-build-dir`.

--keep-build-dir
Don’t remove `pio ci --build-dir` after build process.

-O, --project-option
Pass additional options from “platformio.ini” (Project Configuration File) to `pio project init` command. For example, automatically install dependent libraries `pio ci --project-option="lib_deps=ArduinoJSON"` or ignore specific library `pio ci --project-option="lib_ignore=SomeLib"`.

Note: Use multiple `--project-option` to pass multiple options to “platformio.ini” (Project Configuration File). One option per one argument. For example, `pio ci --project-option="build_unflags =`
-std=gnu++11" --project-option="build_flags = -std=c++14"

-\texttt{v, --verbose}

Shows detailed information when processing environments.
This option can also be set globally using \texttt{force_verbose} setting or by environment variable \texttt{PLATFORMIO_SETTING_FORCE_VERBOSE}.

\section*{Examples}

For the others examples please follow to \textit{Continuous Integration} page.

\texttt{pio debug}

Helper command for \textit{Debugging}.

\section*{Contents}

- \texttt{pio debug}
  - \textit{Usage}
  - \textit{Description}
  - \textit{Options}
  - \textit{Examples}

\section*{Usage}

\begin{verbatim}
pio debug [OPTIONS]
\end{verbatim}

\begin{verbatim}
# A binary shortcut for "pio debug --interface=gdb" command
piodebuggdb [GDB OPTIONS]
\end{verbatim}

\section*{Description}

Prepare PlatformIO project for debugging or launch debug server.

\section*{Options}

-\texttt{e, --environment}

Debug specified environments.
You can also specify which environments should be used for debugging by default using \texttt{default_envs} option from \texttt{"platformio.ini"} (Project Configuration File).

-\texttt{d, --project-dir}

Specify the path to a project directory. By default, \texttt{--project-dir} is equal to a current working directory (CWD).
-c, --project-conf

Process project with a custom “platformio.ini” (Project Configuration File).

--interface

PlatformIO Debugging Interface. Valid values:

- gdb - GDB: The GNU Project Debugger

-v, --verbose

Shows detailed information when processing environments.

This option can also be set globally using force_verbose setting or by environment variable PLATFORMIO_SETTING_FORCE_VERBOSE.

Examples

1. Prepare a project for debugging

```
> pio debug

[Sun Apr 30 01:34:01 2017] Processing mzeropro (platform: atmelsam; debug_extra_cmds: b main.cpp:26; board: mzeropro; framework: arduino)

Verbose mode can be enabled via `-v, --verbose` option
Collected 26 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/mzeropro/src/main.o
Compiling .pio/build/mzeropro/FrameworkArduinoVariant/variant.o
Compiling .pio/build/mzeropro/FrameworkArduino/IPAddress.o
Compiling .pio/build/mzeropro/FrameworkArduino/Print.o
Archiving .pio/build/mzeropro/libFrameworkArduinoVariant.a
Indexing .pio/build/mzeropro/libFrameworkArduinoVariant.a
... Compiling .pio/build/mzeropro/FrameworkArduino/wiring_analog.o
Compiling .pio/build/mzeropro/FrameworkArduino/wiring_digital.o
Compiling .pio/build/mzeropro/FrameworkArduino/wiring_private.o
Compiling .pio/build/mzeropro/FrameworkArduino/wiring_shift.o
Archiving .pio/build/mzeropro/libFrameworkArduino.a
Indexing .pio/build/mzeropro/libFrameworkArduino.a
Linking .pio/build/mzeropro/firmware.elf
Calculating size .pio/build/mzeropro/firmware.elf
Building .pio/build/mzeropro/firmware.bin

text  data  bss  dec  hex  filename
11512  256  1788  13556  34f4  .pio/build/mzeropro/firmware.elf
                       [SUCCESS] Took 7.82 seconds
```

2. Launch GDB instance and load initial configuration per a project

```
> pio debug --interface=gdb -x .pioinit

... Loading section .text, size 0x2c98 lma 0x4000
Loading section .ramfunc, size 0x60 lma 0x6c98
Loading section .data, size 0x100 lma 0x6cf8
```
(continues on next page)
Start address 0x47b0, load size 11768
Transfer rate: 4 KB/sec, 3922 bytes/write.
target halted due to debug-request, current mode: Thread
xPSR: 0x81000000 pc: 0x000028f4 msp: 0x20002c00
target halted due to debug-request, current mode: Thread
xPSR: 0x81000000 pc: 0x000028f4 msp: 0x20002c00
Breakpoint 2 at 0x413a: file src/main.cpp, line 26.

Device Manager CLI

To print all available commands and options use:

```
pio device --help
pio device COMMAND --help
```

**pio device list**

**Contents**

- **pio device list**
  - **Usage**
  - **Description**
  - **Options**
  - **Examples**

**Usage**

```
pio device list [OPTIONS]
```

**Description**

List available devices. Default is set to `--serial` and all available Serial Ports will be shown.

**Options**

- **--serial**
  List available Serial Ports, default.
- **--logical**
  List available logical devices.
- **--mdns**
  List multicast DNS services.
**--json-output**

Return the output in **JSON** format.

**Examples**

1. **Unix OS**

   ```
   > pio device list
   /dev/cu.SLAB_USBtoUART
   ----------
   Hardware ID: USB VID:PID=10c4:ea60 SNR=0001
   Description: CP2102 USB to UART Bridge Controller
   
   /dev/cu.uart-1CFF4676258F4543
   ----------
   Hardware ID: USB VID:PID=451:f432 SNR=1CFF4676258F4543
   Description: Texas Instruments MSP-FET430UIF
   ```

2. **Windows OS**

   ```
   > pio device list
   COM4
   ----------
   Hardware ID: USB VID:PID=0451:F432
   Description: MSP430 Application UART (COM4)

   COM3
   ----------
   Hardware ID: USB VID:PID=10C4:EA60 SNR=0001
   Description: Silicon Labs CP210x USB to UART Bridge (COM3)
   ```

3. **List multicast DNS services and logical devices**

   ```
   > pio device list --mdns --logical
   Multicast DNS Services
   =======================
   PlatformIO._bttremote._tcp.local.
   ---------------------------------
   Type: _bttremote._tcp.local.
   IP: ...
   Port: 62941
   Properties: ...

   Time for PlatformIO._adisk._tcp.local.
   -------------------------------------
   Type: _adisk._tcp.local.
   IP: 192.168.0.1
   Port: 9
   Properties: ...

   PlatformIO._ssh._tcp.local.
   --------------------------
   Type: _ssh._tcp.local.
   IP: ...
   Port: 22
   ```

(continues on next page)
pio device monitor

Contents

- pio device monitor
  - Usage
  - Description
  - Options
  - Filters
  - Capture output to a file
  - Device Monitor Filter API
  - Examples

Usage

pio device monitor [OPTIONS]

Description

This is a console application that provides a small terminal application. It is based on Miniterm and itself does not implement any terminal features such as VT102 compatibility. However it inherits these features from the terminal it is run. For example on GNU/Linux running from an xterm it will support the escape sequences of the xterm. On
Windows the typical console window is dumb and does not support any escapes. When ANSI.sys is loaded it supports some escapes.

Miniterm supports RFC 2217 remote serial ports and raw sockets using URL Handlers such as rfc2217://<host>:<port> respectively socket://<host>:<port> as port argument when invoking.

To control monitor please use these “hot keys”:

- Ctrl+C Quit
- Ctrl+T Menu
- Ctrl+T followed by Ctrl+H Help

**Options**

-p, --port
Port, a number or a device name, or valid URL Handlers.

Can be customized in “platformio.ini” (Project Configuration File) using monitor_port option.

**URL Handlers**

- rfc2217://<host>:<port>[?<option> [&<option> ...]]
- socket://<host>:<port>[?logging={debug|info|warning|error}]
- loop://[?logging={debug|info|warning|error}]
- hwgrep://<regexp>[&skip_busy][&n=N]
- spy://port[?option=value [&option=value]]
- alt://port?class=<classname>

-b, --baud
Set baud rate, default 9600.

Can be customized in “platformio.ini” (Project Configuration File) using monitor_speed option.

--parity
Set parity (None, Even, Odd, Space, Mark), one of [N, E, O, S, M], default N

--rtscts
Enable RTS/CTS flow control, default Off

--xonxoff
Enable software flow control, default Off

--rts
Set initial RTS line state (0 or 1).

Can be customized in “platformio.ini” (Project Configuration File) using monitor_rts option.

--dtr
Set initial DTR line state (0 or 1).

Can be customized in “platformio.ini” (Project Configuration File) using monitor_dtr option.

--echo
Enable local echo, default Off

---encoding
Set the encoding for the serial port (e.g. hexlify, Latin1, UTF-8), default UTF-8.

--filter
Add text transformation. See available filters at Filters.

--eol
End of line mode (CR, LF or CRLF), default CRLF

NEW: Available in Miniterm/PySerial 3.0

--raw
Do not apply any encodings/transformations

--exit-char
ASCII code of special character that is used to exit the application, default 3 (DEC, Ctrl+C).

For example, to use Ctrl+] run pio device monitor --exit-char 29.

--menu-char
ASCII code of special character that is used to control miniterm (menu), default 20 (DEC)

---quiet
Diagnostics: suppress non-error messages, default Off

-d, --project-dir
Specify the path to project directory. By default, --project-dir is equal to current working directory (CWD).

-e, --environment
Process specified environments.

You can also specify which environments should be processed by default using default_envs option from “platformio.ini” (Project Configuration File).

Filters

A list of filters that can be applied for monitor output using pio device monitor --filter or “platformio.ini” (Project Configuration File) and monitor_filters options.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>Remove typical terminal control codes from input</td>
</tr>
<tr>
<td>colorize</td>
<td>Apply different colors for received and echo</td>
</tr>
<tr>
<td>debug</td>
<td>Print what is sent and received</td>
</tr>
<tr>
<td>direct</td>
<td>Do-nothing: forward all data unchanged</td>
</tr>
<tr>
<td>hexlify</td>
<td>Show a hexadecimal representation of the data (code point of each character)</td>
</tr>
<tr>
<td>log2file</td>
<td>Log data to a file “platformio-device-monitor-%date%.log” located in the current working directory</td>
</tr>
<tr>
<td>nocontrol</td>
<td>Remove all control codes, incl. CR+LF</td>
</tr>
<tr>
<td>printable</td>
<td>Show decimal code for all non-ASCII characters and replace most control codes</td>
</tr>
<tr>
<td>time</td>
<td>Add timestamp with milliseconds for each new line</td>
</tr>
<tr>
<td>send_on_enter</td>
<td>Send a text to device on ENTER</td>
</tr>
<tr>
<td>esp32_exception_decoder</td>
<td>Custom filter for Espressif 32 which decodes crash exception</td>
</tr>
<tr>
<td>esp8266_exception_decoder</td>
<td>Custom filter for Espressif 8266 which decodes crash exception</td>
</tr>
</tbody>
</table>

**Capture output to a file**

You need to use a `log2file` filter from *Filters*:

```plaintext
> pio device monitor -f log2file -f default
```

or using “`platformio.ini`” *(Project Configuration File)* and *monitor_filters*

```plaintext
[env:log_output_to_file]
...
platform = ...
monitor_filters = log2file, default
```

**Device Monitor Filter API**

*PlatformIO Core (CLI)* provides an API to extend device monitor with a custom filter declared in “monitor” folder of *Development Platforms*. See examples:

- [https://github.com/platformio/platform-espressif32/tree/develop/monitor](https://github.com/platformio/platform-espressif32/tree/develop/monitor)
- [https://github.com/platformio/platform-espressif8266/tree/develop/monitor](https://github.com/platformio/platform-espressif8266/tree/develop/monitor)

**Examples**

1. Show available options for `monitor`

```plaintext
> pio device monitor --help
Usage: pio device monitor [OPTIONS]

Options:
    -p, --port TEXT          Port, a number or a device name
    -b, --baud INTEGER       Set baud rate, default=9600
    --parity [N|E|O|S|M]     Set parity, default=N
    --rtscts                 Enable RTS/CTS flow control, default=Off
    --xonxoff                Enable software flow control, default=Off
    --rts [0|1]              Set initial RTS line state, default=0
```

(continues on next page)
2. Communicate with serial device and print help inside terminal

> pio device monitor

--- Available ports:
--- /dev/cu.Bluetooth-Incoming-Port n/a
--- /dev/cu.Bluetooth-Modem n/a
--- /dev/cu.SLAB_USBtoUART CP2102 USB to UART Bridge Controller
--- /dev/cu.obd2ecu-SPPDev n/a
Enter port name:/dev/cu.SLAB_USBtoUART

--- Miniterm on /dev/cu.SLAB_USBtoUART: 9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Hello PlatformIO!
---
--- Ctrl+] Exit program
--- Ctrl+T Menu escape key, followed by:
--- Menu keys:
--- Ctrl+T Send the menu character itself to remote
--- Ctrl+] Send the exit character itself to remote
--- Ctrl+I Show info
--- Ctrl+U Upload file (prompt will be shown)
--- Toggles:
--- Ctrl+R RTS Ctrl+E local echo
--- Ctrl+D DTR Ctrl+B BREAK
--- Ctrl+L line feed Ctrl+A Cycle repr mode
---
--- Port settings (Ctrl+T followed by the following):
--- p change port
--- 7 8 set data bits
--- n e o s m change parity (None, Even, Odd, Space, Mark)
--- 1 2 3 set stop bits (1, 2, 1.5)
--- b change baud rate
--- x X disable/enable software flow control
--- r R disable/enable hardware flow control
--- exit ---

pio home

Helper command for PlatformIO Home.
Contents

- pio home
  - Usage
  - Description
  - Options
  - Examples

Usage

pio home

Description

Launch PlatformIO Home Web-server.

Options

--port
Web-server HTTP port, default is 8008.

--host
Web-server HTTP host, default is 127.0.0.1. You can open PlatformIO Home for inbound connections using host 0.0.0.0.

--no-open
Do not automatically open PlatformIO Home in a system Web-browser.

--shutdown-timeout
Automatically shutdown server on timeout (in seconds) when no clients are connected. Default is 0 which means never auto shutdown.

Examples

> pio home

___I__
/\-__\ PlatformIO Home
/ \_-__\ [][[]] | http://127.0.0.1:8008
|__|___|_________________________

Open PlatformIO Home in your browser by this URL => http://127.0.0.1:8008
PlatformIO Home has been started. Press Ctrl+C to shutdown.
Library Manager CLI

Usage

# To print all available commands and options use
pio lib --help
pio lib COMMAND --help

Options

-d, --storage-dir
Manage custom library storage. It can be used later for the `lib_extra_dirs` option from "platformio.ini" (Project Configuration File). Multiple options are allowed.

-g, --global
Manage global PlatformIO’s library storage ("core_dir/lib") where Library Dependency Finder (LDF) will look for dependencies by default.

-e, --environment
Manage libraries for the specific project build environments declared in "platformio.ini" (Project Configuration File). Works for --storage-dir which is valid PlatformIO project.

Demo

Commands

pio lib builtin

Contents

* pio lib builtin
  * Usage
  * Description
  * Options
  * Examples

Usage

pio lib builtin [OPTIONS]
Description

List built-in libraries based on installed Development Platforms and their frameworks, SDKs, etc.

Options

--storage
List libraries from specified storages. For example, framework-arduinoavr.

--json-output
Return the output in JSON format

Examples

> pio lib builtin
framework-arduinoavr
********************
Bridge
======
Enables the communication between the Linux processor and the microcontroller. For Arduino/Genuino Yún, Yún Shield and TRE only.

Version: 1.6.1
Homepage: http://www.arduino.cc/en/Reference/YunBridgeLibrary
Keywords: communication
Compatible frameworks: arduino
Compatible platforms: *
Authors: Arduino

EEPROM
======
Enables reading and writing to the permanent board storage.

Version: 2.0
Homepage: http://www.arduino.cc/en/Reference/EEPROM
Keywords: data, storage
Compatible frameworks: arduino
Compatible platforms: atmelavr
Authors: Arduino, Christopher Andrews

framework-arduinosam
********************
Audio
====
Allows playing audio files from an SD card. For Arduino DUE only.

Version: 1.0

(continues on next page)
SPI
===
Enables the communication with devices that use the Serial Peripheral Interface (SPI) → Bus. For all Arduino boards, BUT Arduino DUE.

Version: 1.0
Keywords: signal, input, output
Compatible frameworks: arduino
Compatible platforms: atmelsam
Authors: Hristo Gochkov

framework-arduinoespressif8266
******************************

ArduinoOTA
==========
Enables Over The Air upgrades, via wifi and espota.py UDP request/TCP download.

Version: 1.0
Keywords: communication
Compatible frameworks: arduino
Compatible platforms: espressif8266
Authors: Ivan Grokhokov and Miguel Angel Ajo

DNSServer
========= A simple DNS server for ESP8266.

Version: 1.1.0
Keywords: communication
Compatible frameworks: arduino
Compatible platforms: espressif8266
Authors: Kristijan Novoselić

Adafruit NeoPixel
=================
Arduino library for controlling single-wire-based LED pixels and strip.
Version: 1.0.3
Homepage: https://github.com/adafruit/Adafruit_NeoPixel
Keywords: display
Compatible frameworks: arduino
Compatible platforms: *
Authors: Adafruit

CurieBLE
========
Library to manage the Bluetooth Low Energy module with Curie Core boards.

Version: 1.0
Keywords: communication
Compatible frameworks: arduino
Compatible platforms: intel_arc32
Authors: Emutex

CurieEEPROM
============
Enables reading and writing to OTP flash area of Curie

Version: 1.0
Homepage: http://www.arduino.cc/en/Reference/EEPROM
Keywords: data, storage
Compatible frameworks: arduino
Compatible platforms: intel_arc32
Authors: Intel

framework-arduinomicrochippic32
********************************

Firmata
=======
Enables the communication with computer apps using a standard serial protocol. For all Arduino boards.

Version: 2.4.4
Homepage: https://github.com/firmata/arduino
Keywords: device, control
Compatible frameworks: arduino
Compatible platforms: *
Authors: Firmata Developers

framework-arduinoteensy
************************

Adafruit CC3000 Library
************************
Library code for Adafruit's CC3000 WiFi breakouts.

Version: 1.0.1
Homepage: https://github.com/adafruit/Adafruit_CC3000_Library
Keywords: communication
Compatible frameworks: arduino
Compatible platforms: *
Authors: Adafruit

framework-energiamsp430
************************

AIR430BoostEuropeETSI
=====================
Library for the CC110L Sub-1GHz radio BoosterPack for use in Europe

Version: 1.0.0
Homepage: http://energia.nu/reference/libraries/
Keywords: communication
Compatible frameworks: arduino
Compatible platforms:
Authors: Energia

framework-energiativa
************************
aJson
======
An Arduino library to enable JSON processing with Arduino

Keywords: json, rest, http, web
Compatible frameworks: arduino
Compatible platforms: atmelavr

pio lib install

Contents

- pio lib install
  - Usage
  - Description
  - Storage Options
  - Options
  - Version control
    - Git
    - Mercurial
    - Subversion
  - Examples
Usage

```shell
pio lib [STORAGE_OPTIONS] install [OPTIONS] [LIBRARY...]  
```

# RECOMMENDED
# install all project dependencies declared via "lib_deps"
# (run it from a project root where is located "platformio.ini")
```shell
pio lib install [OPTIONS]
```

# install project dependent library
# (run it from a project root where is located "platformio.ini")
```shell
pio lib install [OPTIONS] [LIBRARY...]
```

# install dependencies for the specific project environment
# (run it from a project root where is located "platformio.ini")
```shell
pio lib -e myenv install [OPTIONS] [LIBRARY...]
pio lib -d /path/to/platformio/project -e myenv install [OPTIONS] [LIBRARY...]
```

# install to global storage (NOT RECOMMENDED)
```shell
pio lib --global install [OPTIONS] [LIBRARY...]
PIO lib -g install [OPTIONS] [LIBRARY...]
```

# install to custom storage
```shell
pio lib --storage-dir /path/to/dir install [OPTIONS] [LIBRARY...]
pio lib -d /path/to/dir1 -d /path/to/dir2 install [OPTIONS] [LIBRARY...]
```

# [LIBRARY...] forms
```shell
pio lib [STORAGE_OPTIONS] install (with no args, install project dependencies from "lib_deps")
pio lib [STORAGE_OPTIONS] install <ownername/name>
pio lib [STORAGE_OPTIONS] install <ownername/name>@<version>
pio lib [STORAGE_OPTIONS] install <ownername/name>@<version range>
pio lib [STORAGE_OPTIONS] install <zip or tarball url>
pio lib [STORAGE_OPTIONS] install file://<zip or tarball file>
pio lib [STORAGE_OPTIONS] install file://<folder>
pio lib [STORAGE_OPTIONS] install <repository>
pio lib [STORAGE_OPTIONS] install <repository#tag> ("tag" can be commit, branch or tag)
```

---

**Warning:** If some libraries are not visible in *PlatformIO IDE* and Code Completion or Code Linting does not work properly, please perform

- **VSCode:** “Menu: View > Command Palette… > PlatformIO: Rebuild C/C++ Project Index”

---

**Description**

Install a library, and any libraries that it depends on using:

1. PlatformIO Library Registry
2. Custom folder, repository or archive.

The version supports **Semantic Versioning** (**<major>**.<**minor**>.<**patch**>) and can take any of the following forms:
• ^1.2.3 - any compatible version (new functionality in a backwards compatible manner and patches are allowed, 1.x.x). **RECOMMENDED**
• ~1.2.3 - any version with the same major and minor versions, and an equal or greater patch version
• >1.2.3 - any version greater than 1.2.3. >=, <, and <= are also possible
• >0.1.0, !=0.2.0, <0.3.0 - any version greater than 0.1.0, not equal to 0.2.0 and less than 0.3.0
• 1.2.3 - an exact version number. Use only this exact version.

PlatformIO supports installing from local directory or archive. You need to use `file://` prefix before local path.

- `file://local/path/to/the/platform/dir`
- `file://local/path/to/the/platform.zip`
- `file://local/path/to/the/platform.tar.gz`

### Storage Options

See base options for *Library Manager CLI*.

### Options

---save / --no-save

Save installed libraries into the “platformio.ini” (*Project Configuration File*) dependency list (*lib_deps*). Default value is to save.

You can save libraries for the specific project environment using `-e, --environment` option from `pio lib` command. For example, `pio lib -e myenv install [LIBRARY...]`.

-s, --silent

Suppress progress reporting.

-f, --force

Reinstall library if it is already installed.

### Version control

PlatformIO supports installing from Git, Mercurial and Subversion, and detects the type of VCS using url prefixes: “git+”, “hg+”, or “svn+”.

**Note:** PlatformIO requires a working VCS command on your path: `git`, `hg` or `svn`.

### Git

The supported schemes are: `git`, `git+https` and `git+ssh`. Here are the supported forms:

- `https://github.com/user/library.git`
- `git+git://git.server.org/my-library`
- `git+https://git.server.org/my-library`
- `git+ssh://git.server.org/my-library`
- `git+ssh://user@git.server.org/my-library`
- `[user@]host.xz:path/to/repo.git`

Passing branch names, a commit hash or a tag name is possible like so:
- `https://github.com/user/library.git#master`
- `git+git://git.server.org/my-library#master`
- `git+https://git.server.org/my-library#v1.0`
- `git+ssh://git.server.org/my-library#7846d8ad52f983f2f2887bdc0f073fe9755a806d`

**Mercurial**

The supported schemes are: `hg+http`, `hg+https` and `hg+ssh`. Here are the supported forms:
- `https://developer.mbed.org/users/user/code/library/` (install ARM mbed library)
- `hg+hg://hg.server.org/my-library`
- `hg+https://hg.server.org/my-library`
- `hg+ssh://hg.server.org/my-library`

Passing branch names, a commit hash or a tag name is possible like so:
- `hg+hg://hg.server.org/my-library#master`
- `hg+https://hg.server.org/my-library#v1.0`
- `hg+ssh://hg.server.org/my-library#4cfe2fa00668`

**Subversion**

The supported schemes are: `svn`, `svn+svn`, `svn+http`, `svn+https` and `svn+ssh`. Here are the supported forms:
- `svn+svn://svn.server.org/my-library`
- `svn+https://svn.server.org/my-library`
- `svn+ssh://svn.server.org/my-library`

You can also give specific revisions to an SVN URL, like so:
- `svn+svn://svn.server.org/my-library#13`

**Examples**

1. Install the latest version of library to a global storage using ID or NAME

```
> pio lib -g install 4
Library Storage: /storage/dir/...
LibraryManager: Installing id=4
Downloading [####################################] 100%
Unpacking [####################################] 100%
```

(continues on next page)
IRremote @ 2.2.1 has been successfully installed!

# repeat command with name
> pio lib -g install IRRemote

Library Storage: /storage/dir/...
Looking for IRRemote library in registry
Found: https://platformio.org/lib/show/4/IRremote
LibraryManager: Installing id=4
IRremote @ 2.2.1 is already installed

2. Install specified version of a library to a global storage

> pio lib -g install ArduinoJson@5.6.7

Library Storage: /storage/dir/...
Looking for ArduinoJson library in registry
Found: https://platformio.org/lib/show/64/ArduinoJson
LibraryManager: Installing id=64 @ 5.6.7
Downloading [####################################] 100%
Unpacking [####################################] 100%
ArduinoJson @ 5.6.7 has been successfully installed!

3. Install library with dependencies to custom storage

> pio lib --storage-dir /my/storage/dir install DallasTemperature

Library Storage: /my/storage/dir
Looking for DallasTemperature library in registry
Found: https://platformio.org/lib/show/54/DallasTemperature
LibraryManager: Installing id=54
Downloading [####################################] 100%
Unpacking [####################################] 100%
DallasTemperature @ 3.7.7 has been successfully installed!
Installing dependencies
Looking for OneWire library in registry
Found: https://platformio.org/lib/show/1/OneWire
LibraryManager: Installing id=1
Downloading [####################################] 100%
Unpacking [####################################] 100%
OneWire @ 8fd2ebfecd has been successfully installed!

4. Install ARM mbed library to the global storage

> pio lib -g install https://developer.mbed.org/users/simon/code/TextLCD/

Library Storage: /storage/dir/...
LibraryManager: Installing TextLCD
Mercurial Distributed SCM (version 3.8.4)
(see https://mercurial-scm.org for more information)

Copyright (C) 2005-2016 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
requesting all changes
adding changesets
adding manifests

(continues on next page)
adding file changes
added 9 changesets with 18 changes to 6 files
updating to branch default
2 files updated, 0 files merged, 0 files removed, 0 files unresolved
TextLCD @ 308d188a2d3a has been successfully installed!

5. Install from archive using URL

> pio lib -g install https://github.com/adafruit/DHT-sensor-library/archive/master.
    → zip

Library Storage: /storage/dir/...
LibraryManager: Installing master
Downloading [####################################] 100%
Unpacking [####################################] 100%
DHT sensor library @ 1.2.3 has been successfully installed!

pio lib list

Contents

- pio lib list
  - Usage
  - Description
  - Storage Options
  - Options
  - Examples

Usage

pio lib [STORAGE_OPTIONS] list [OPTIONS]

# list project dependent libraries
# (run it from a project root where is located "platformio.ini")
pio lib list [OPTIONS]

# list libraries from global storage
pio lib --global list [OPTIONS]
pio lib -g list [OPTIONS]

# list libraries from custom storage
pio lib --storage-dir /path/to/dir list [OPTIONS]
pio lib -d /path/to/dir list [OPTIONS]

Description

List installed libraries
Storage Options

See base options for *Library Manager CLI*.

Options

`--json-output`

Return the output in **JSON** format

Examples

```bash
> pio lib -g list

Library Storage: /storage/dir/...

Adafruit Unified Sensor
-----------------------
#ID: 31
Required **for all** Adafruit Unified Sensor based libraries.

Version: **1.0.2**
Keywords: sensors
Compatible frameworks: arduino
Compatible platforms: atmelavr, atmelsam, espressif8266, intel_arc32, microchippic32, ...
  → nordicnrf51, teensy, timsp430
Authors: Adafruit

ArduinoJson
============
#ID: 64
An elegant and efficient JSON library **for** embedded systems

Version: **5.8.0**
Keywords: web, json, http, rest
Compatible frameworks: arduino
Compatible platforms: atmelavr, atmelsam, espressif8266, intel_arc32, microchippic32, ...
  → nordicnrf51, teensy, timsp430
Authors: Benoit Blanchon

ArduinoJson
============
#ID: 64
An elegant and efficient JSON library **for** embedded systems

Version: **5.6.7**
Keywords: web, json, http, rest
Compatible frameworks: arduino
Compatible platforms: atmelavr, atmelsam, espressif8266, intel_arc32, microchippic32, ...
  → nordicnrf51, teensy, timsp430
Authors: Benoit Blanchon

ArduinoJson
============
#ID: 64
```
An elegant and efficient JSON library for embedded systems

Version: 5.7.2
Keywords: web, json, http, rest
Compatible frameworks: arduino
Compatible platforms: atmelavr, atmelsam, espressif8266, intel_arc32, microchippic32, nordicnrf51, teensy, timsp430
Authors: Benoit Blanchon

Blynk
======
#ID: 415

Build a smartphone app for your project in minutes. Blynk allows creating IoT solutions easily. It supports WiFi, BLE, Bluetooth, Ethernet, GSM, USB, Serial. Works with many boards like ESP8266, ESP32, Arduino UNO, Nano, Due, Mega, Zero, MKR1000, Yun, Raspberry Pi, Particle, Energia, ARM mbed, Intel Edison/Galileo/Joule, BBC micro:bit, DFRobot, RedBearLab, Microduino, LinkIt ONE ...

Version: 0.4.3
Homepage: http://blynk.cc
Keywords: control, gprs, protocol, communication, app, bluetooth, serial, cloud, web, us, m2m, ble, 3g, smartphone, http, iot, device, sensors, data, esp8266, mobile, wifi, ethernet, gsm
Compatible frameworks: energia, wiringpi, arduino
Compatible platforms: atmelavr, atmelsam, espressif8266, intel_arc32, linux_arm, microchippic32, nordicnrf51, teensy, timsp430, titiva
Authors: Volodymyr Shymanskyy

Bounce2
=======
#ID: 1106

Debouncing library for Arduino or Wiring

Version: 2.1
Keywords: input, signal, output, bounce
Compatible frameworks: arduino
Compatible platforms: atmelavr, atmelsam, espressif8266, intel_arc32, microchippic32, nordicnrf51, teensy, timsp430
Authors: Thomas O Fredericks

Homie
=====
#ID: 555

ESP8266 framework for Homie, a lightweight MQTT convention for the IoT

Version: 1.5.0
Keywords: home, mqtt, iot, esp8266, automation
Compatible frameworks: arduino
Compatible platforms: espressif8266
Authors: Marvin Roger

JustWifi
=======
#ID: 1282

Wifi Manager for ESP8266 that supports multiple wifi networks and scan for strongest signal
LiquidCrystal
==============

**Version:** 1.1.1
**License:** GPL-3.0
**Keywords:** manager, wifi, scan
**Compatible frameworks:** arduino
**Compatible platforms:** espressif8266
**Authors:** Xose Perez

LiquidCrystal Library is faster **and** extensible, compatible **with** the original_

---

**Version:** 1.3.4
**Keywords:** lcd, hd44780
**Compatible frameworks:** arduino
**Compatible platforms:** atmelavr
**Authors:** F Malpartida

TextLCD
=======

**Version:** 308d188a2d3a
**Keywords:** uncategorized

**Time**

**Version:** 1.5
**Homepage:** http://playground.arduino.cc/Code/Time
**Keywords:** week, rtc, hour, year, month, second, time, date, day, minute
**Compatible frameworks:** arduino
**Compatible platforms:**
**Authors:** Michael Margolis, Paul Stoffregen

Timezone
========

**Version:** 510ae2f6b6
**Keywords:** zone, time
**Compatible frameworks:** arduino
**Compatible platforms:** atmelavr
**Authors:** Jack Christensen

U8g2
====

**Version:** 942
**Keywords:** Monochrome LCD, OLED and eInk Library. Display controller: SSD1305, SSD1306, SSD1322, SSD1325, SSD1327, SSD1606, SH1106, T6963, RA8835, LC7981, PCD8544, PCF8812, UC1604, UC1608, UC1610, UC1611, UC1701, ST7565, ST7567, NT7534, ST7920, LD7032, KS0108.
**Interfaces:** I2C, SPI, Parallel.
Version: **2.11.4**  
Homepage: [https://github.com/olikraus/u8g2](https://github.com/olikraus/u8g2)  
Keywords: display  
Compatible platforms: arduino  
Authors: oliver  

**USB-Host-Shield-20**  
---

**ID: 59**  
Revision **2.0** of MAX3421E-based USB Host Shield Library  

**Version:** **1.2.1**  
License: **GPL-2.0**  
Keywords: usb, spp, mass storage, pl2303, acm, ftdi, xbox, host, hid, wii, buzz, ps3, bluetooth, adk, ps4  
Compatible frameworks: spl, arduino  
Compatible platforms: atmelavr, atmelsam, teensy, nordicnrf51, ststm32  
Authors: Oleg Mazurov, Alexei Glushchenko, Kristian Lauszus, Andrew Kroll

---

**pio lib register**

**DEPRECATED.**

Please use `pio package publish`.

**pio lib search**

---

**Contents**

- **pio lib search**
  - **Usage**
  - **Description**
  - **Options**
  - **Examples**

**Usage**

```
pio lib search [OPTIONS] [QUERY]
```

**Description**

Search for library in PlatformIO Library Registry by `library.json` fields in the boolean mode.
The boolean search capability supports the following operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>A leading or trailing plus sign indicates that this word must be present in library fields (see above) that is returned.</td>
</tr>
<tr>
<td>-</td>
<td>A leading or trailing minus sign indicates that this word must not be present in any of the libraries that are returned.</td>
</tr>
<tr>
<td>(no operator)</td>
<td>By default (when neither + nor - is specified), the word is optional, but the libraries that contain it are rated higher.</td>
</tr>
<tr>
<td>&gt; &lt;</td>
<td>These two operators are used to change a word’s contribution to the relevance value that is assigned to a library. The &gt; operator increases the contribution and the &lt; operator decreases it.</td>
</tr>
<tr>
<td>( )</td>
<td>Parentheses group words into subexpressions. Parenthesized groups can be nested.</td>
</tr>
<tr>
<td>~</td>
<td>A leading tilde acts as a negation operator, causing the word’s contribution to the library’s relevance to be negative. This is useful for marking “noise” words. A library containing such a word is rated lower than others, but is not excluded altogether, as it would be with the – operator.</td>
</tr>
<tr>
<td>*</td>
<td>The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it is appended to the word to be affected. Words match if they begin with the word preceding the + operator.</td>
</tr>
<tr>
<td>&quot;</td>
<td>A phrase that is enclosed within double quote (&quot;) characters matches only libraries that contain the phrase literally, as it was typed.</td>
</tr>
</tbody>
</table>

For more detail information please go to MySQL Boolean Full-Text Searches.

**Options**

---id
Filter libraries by registry ID

--n, --name
Filter libraries by specified name (strict search)

--a, --author
Filter libraries by specified author

--k, --keyword
Filter libraries by specified keyword

--f, --framework
Filter libraries by specified framework

--p, --platform
Filter libraries by specified keyword

--i, --header
Filter libraries by header file (include)

For example, pio lib search --header "OneWire.h"

--json-output
Return the output in JSON format

--page
Examples

1. List all libraries

```bash
> pio lib search

Found N libraries:

ArduinoJson
=============
#ID: 64
An elegant and efficient JSON library for embedded systems
Keywords: web, json, http, rest
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 8266, Intel ARC32, Microchip, PIC32, Nordic nRF51, Teensy, TI MSP430
Authors: Benoit Blanchon

DHT sensor library
==================
#ID: 19
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors
Keywords: unified, dht, sensor, temperature, humidity
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Adafruit Industries

PubSubClient
============
#ID: 89
A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1...
Keywords: ethernet, mqtt, iot, m2m
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 8266, Intel ARC32, Microchip, PIC32, Nordic nRF51, Teensy, TI MSP430
Authors: Nick O'Leary

ESPAsyncWebServer
==================
#ID: 306
Asynchronous HTTP and WebSocket Server Library for ESP8266 and ESP32
Keywords: async, websocket, http, webserver
Compatible frameworks: Arduino
Compatible platforms: Espressif 8266
Authors: Hristo Gochkov
```
(continues on next page)
2. Search for 1-Wire libraries

```
> pio lib search "1-wire"
```

```
Found N libraries:

DS1820
======
#ID: 196
Dallas / Maxim DS1820 1-Wire library. For communication with multiple DS1820 on a
--single 1-Wire bus. Also supports DS18S20 and DS18B20.
Keywords: ds18s20, 1-wire, ds1820, ds18b20
Compatible frameworks: mbed
Compatible platforms: Freescale Kinetis, Nordic nRF51, NXP LPC, ST STM32, Teensy
Authors: Michael Hagberg
```

```
OneWire
=======
#ID: 1
Control 1-Wire protocol (DS18S20, DS18B20, DS2408 and etc)
Keywords: onewire, temperature, bus, 1-wire, ibutton, sensor
Compatible frameworks: Arduino
Compatible platforms:
Authors: Paul Stoffregen, Jim Studt, Tom Pollard, Derek Yerger, Josh Larios, Robin
--James, Glenn Trewitt, Jason Dangel, Guillermo Lovato, Ken Butcher, Mark Tillotson,
--Bertrik Sikken, Scott Roberts
```

Show next libraries? [y/N]: ...

3. Search for Arduino-based “I2C” libraries

```
> pio lib search "i2c" --framework="arduino"
```

```
Found N libraries:

I2Cdevlib-AK8975
================
#ID: 10
AK8975 is 3-axis electronic compass IC with high sensitive Hall sensor technology
Keywords: i2c, i2cdevlib, sensor, compass
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Jeff Rowberg
```

```
I2Cdevlib-Core
==============
#ID: 11
The I2C Device Library (I2Cdevlib) is a collection of uniform and well-documented,
--classes to provide simple and intuitive interfaces to I2C devices.
```

(continues on next page)
Keywords: i2cdevlib, i2c
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Jeff Rowberg

Adafruit 9DOF Library
=====================
#ID: 14
Unified sensor driver for the Adafruit 9DOF Breakout (L3GD20 / LSM303)

Keywords: magnetometer, unified, accelerometer, spi, compass, i2c, sensor, gyroscope
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Adafruit Industries

Show next libraries? [y/N]: ...

4. Search for libraries by “web” and “http” keywords.

> pio lib search --keyword="web" --keyword="http"

Found N libraries:

ArduinoJson
============
#ID: 64
An elegant and efficient JSON library for embedded systems

Keywords: web, json, http, rest
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 8266, Intel ARC32, Microchip
→PIC32, Nordic nRF51, Teensy, TI MSP430
Authors: Benoit Blanchon

ESPAsyncWebServer
=================
#ID: 306
Asynchronous HTTP and WebSocket Server Library for ESP8266 and ESP32

Keywords: async, websocket, http, webserver
Compatible frameworks: Arduino
Compatible platforms: Espressif 8266
Authors: Hristo Gochkov

ESP8266wifi
============
#ID: 1101
ESP8266 Arduino library with built in reconnect functionality

Keywords: web, http, wifi, server, client, wi-fi
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Jonas Ekstrand

Blynk
=====

(continues on next page)
Build a smartphone app for your project in minutes. Blynk allows creating IoT solutions easily. It supports WiFi, BLE, Bluetooth, Ethernet, GSM, USB, Serial. Works with many boards like ESP8266, ESP32, Arduino UNO, Nano, Due, Mega, Zero, MKR100, Yun,...

Keywords: control, gprs, protocol, communication, app, bluetooth, serial, cloud, web, usb, m2m, ble, 3G, smartphone, http, iot, device, sensors, data, esp8266, mobile, wifi, ethernet, gsm

Compatible frameworks: Arduino, Energia, WiringPi
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 8266, Intel ARC32, Linux ARM, Microchip PIC32, Nordic nRF51, Teensy, TI MSP430, TI Tiva
Authors: Volodymyr Shymanskyy

Show next libraries? [y/N]:

5. Search for libraries by “Adafruit Industries” author

> pio lib search --author="Adafruit Industries"

Found N libraries:

DHT sensor library
==================
#ID: 19
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors

Keywords: unified, dht, sensor, temperature, humidity
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Adafruit Industries

Adafruit DHT Unified
=====================
#ID: 18
Unified sensor library for DHT (DHT11, DHT22 and etc) temperature and humidity sensors

Keywords: unified, dht, sensor, temperature, humidity
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Adafruit Industries

Show next libraries? [y/N]:

6. Search for libraries which are compatible with Dallas temperature sensors like DS18B20, DS18S20 and etc.

> pio lib search "DS*"

Found N libraries:

DS1820
======
#ID: 196
Dallas / Maxim DS1820 1-Wire library. For communication with multiple DS1820 on a single 1-Wire bus. Also supports DS18S20 and DS18B20.
Keywords: ds18s20, 1-wire, ds1820, ds18b20
Compatible frameworks: mbed
Compatible platforms: Freescale Kinetis, Nordic nRF51, NXP LPC, ST STM32, Teensy
Authors: Michael Hagberg

I2Cdevlib-DS1307
=============
#ID: 99
The DS1307 serial real-time clock (RTC) is a low-power, full binary-coded decimal-
→(BCD) clock/calendar plus 56 bytes of NV SRAM

Keywords: i2cdevlib, clock, i2c, rtc, time
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Jeff Rowberg

Show next libraries? [y/N]: ...

7. Search for Energia-based *nRF24* or *HttpClient* libraries. The search query that is described below can be
interpreted like energia nRF24 OR energia HttpClient

> pio lib search "+(nRF24 HttpClient)" --framework="arduino"

Found N libraries:

RadioHead
=========
#ID: 124
The RadioHead Packet Radio library which provides a complete object-oriented library,
→for sending and receiving packetized messages via RF22/24/26/27/69, Si4460/4461/
→4463/4464, nRF24/nRF905, SX1276/77/78, RFM95/96/97/98 and etc.

Keywords: rf, radio, wireless
Compatible frameworks: Arduino, Energia
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 32, Espressif 8266, Infineon,
→XMC, Intel ARC32, Kendryte K210, Microchip PIC32, Nordic nRF51, Nordic nRF52, ST,
→STM32, ST STM8, Teensy, TI MSP430, TI Tiva
Authors: Mike McCauley

ArduinoHttpClient
=================
#ID: 798
[EXPERIMENTAL] Easily interact with web servers from Arduino, using HTTP and WebSocket
→'s.

Keywords: communication
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 32, Espressif 8266, Intel ARC32,
→Microchip PIC32, Nordic nRF51, Nordic nRF52, ST STM32, ST STM8, Teensy, TI MSP430
Authors: Arduino

HttpClient
==========
#ID: 66
(continues on next page)
Library to easily make HTTP GET, POST and PUT requests to a web server.

Keywords: communication
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 32, Espressif 8266, Intel ARC32, Microchip PIC32, Nordic nRF51, Nordic nRF52, ST STM32, Teensy, TI MSP430
Authors: Adrian McEwen

Show next libraries? [y/N]:
...

8. Search for the all sensor libraries excluding temperature.

> pio lib search "sensor -temperature"

Found N libraries:

SparkFun VL6180 Sensor
======================
#ID: 407
The VL6180 combines an IR emitter, a range sensor, and an ambient light sensor together for you to easily use and communicate with via an I2C interface.

Keywords: sensors
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR, Atmel SAM, Espressif 8266, Intel ARC32, Microchip PIC32, Nordic nRF51, Teensy, TI MSP430
Authors: Casey Kuhns@SparkFun, SparkFun Electronics

I2Cdevlib-AK8975
================
#ID: 10
AK8975 is 3-axis electronic compass IC with high sensitive Hall sensor technology

Keywords: i2c, i2cdevlib, sensor, compass
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Jeff Rowberg

Adafruit 9DOF Library
=====================
#ID: 14
Unified sensor driver for the Adafruit 9DOF Breakout (L3GD20 / LSM303)

Keywords: magnetometer, unified, accelerometer, spi, compass, i2c, sensor, gyroscope
Compatible frameworks: Arduino
Compatible platforms: Atmel AVR
Authors: Adafruit Industries

Show next libraries? [y/N]:
...

pio lib show
Contents

- pio lib show
  - Usage
  - Description
  - Options
  - Examples

Usage

pio lib show [LIBRARY]

Description

Show detailed info about a library using PlatformIO Library Registry.

The possible values for [LIBRARY]:

- Library ID from Registry (preferred)
- Library Name

Options

--json-output

Return the output in JSON format

Examples

> pio lib show OneWire

PubSubClient
===========
ID: 89
A client library for MQTT messaging. MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1...

Version: 2.6, released 10 months ago
--json
Homepage: http://pubsubclient.knolleary.net
Repository: https://github.com/knolleary/pubsubclient.git

Authors
-------
Nick O'Leary https://github.com/knolleary

(continues on next page)
Keywords
--------
ethernet
mqtt
iot
m2m

Compatible frameworks
----------------------
Arduino

Compatible platforms
---------------------
Atmel AVR
Atmel SAM
Espressif **8266**
Intel ARC32
Microchip PIC32
Nordic nRF51
Teensy
TI MSP430

Headers
-------
PubSubClient.h

Examples
--------
http://dl.platformio.org/libraries/examples/0/89/mqtt_auth.ino
http://dl.platformio.org/libraries/examples/0/89/mqtt_basic.ino
http://dl.platformio.org/libraries/examples/0/89/mqtt_esp8266.ino
http://dl.platformio.org/libraries/examples/0/89/mqtt_publish_in_callback.ino
http://dl.platformio.org/libraries/examples/0/89/mqtt_reconnect_nonblocking.ino
http://dl.platformio.org/libraries/examples/0/89/mqtt_stream.ino

Versions
--------
**2.6**, released **10** months ago

Downloads
---------
Today: **25**
Week: **120**
Month: **462**
Usage

```
pio lib stats
```

Description

Show PlatformIO Library Registry statistics:

- Recently updated
- Recently added
- Recent keywords
- Popular keywords
- Featured: Today
- Featured: Week
- Featured: Month

This information is the same that is shown on this page:

- https://platformio.org/lib

Options

```
--json-output
```

Return the output in JSON format

Examples

```
RECENTLY UPDATED
***************
Name Date Url
- GroveEncoder 12 hours ago https://platformio.org/lib/show/1382/
- RF24G 12 hours ago https://platformio.org/lib/show/1381/
- Sim800L Library Revised 12 hours ago https://platformio.org/lib/show/1380/
- ArduinoSTL 12 hours ago https://platformio.org/lib/show/750/
```

(continues on next page)
<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroveEncoder</td>
<td>12 hours ago</td>
<td><a href="https://platformio.org/lib/show/1382/">https://platformio.org/lib/show/1382/</a></td>
</tr>
<tr>
<td>RF24G</td>
<td>12 hours ago</td>
<td><a href="https://platformio.org/lib/show/1381/">https://platformio.org/lib/show/1381/</a></td>
</tr>
<tr>
<td>Sim800L Library Revised</td>
<td>12 hours ago</td>
<td><a href="https://platformio.org/lib/show/1380/">https://platformio.org/lib/show/1380/</a></td>
</tr>
<tr>
<td>DS3231</td>
<td>a day ago</td>
<td><a href="https://platformio.org/lib/show/1379/">https://platformio.org/lib/show/1379/</a></td>
</tr>
<tr>
<td>ArduboyPlaytune</td>
<td>4 days ago</td>
<td><a href="https://platformio.org/lib/show/1378/">https://platformio.org/lib/show/1378/</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>cobs</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Acobs">https://platformio.org/lib/search?query=keyword%3Acobs</a></td>
</tr>
<tr>
<td>packet</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Apacket">https://platformio.org/lib/search?query=keyword%3Apacket</a></td>
</tr>
<tr>
<td>framing</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Aframing">https://platformio.org/lib/search?query=keyword%3Aframing</a></td>
</tr>
<tr>
<td>3g</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3A3g">https://platformio.org/lib/search?query=keyword%3A3g</a></td>
</tr>
<tr>
<td>tdd</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Atdd">https://platformio.org/lib/search?query=keyword%3Atdd</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>display</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Adisplay">https://platformio.org/lib/search?query=keyword%3Adisplay</a></td>
</tr>
<tr>
<td>1cd</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3A1cd">https://platformio.org/lib/search?query=keyword%3A1cd</a></td>
</tr>
<tr>
<td>sensors</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Asensors">https://platformio.org/lib/search?query=keyword%3Asensors</a></td>
</tr>
<tr>
<td>graphics</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3AGraphics">https://platformio.org/lib/search?query=keyword%3AGraphics</a></td>
</tr>
<tr>
<td>communication</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Acommunication">https://platformio.org/lib/search?query=keyword%3Acommunication</a></td>
</tr>
<tr>
<td>oled</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Aoled">https://platformio.org/lib/search?query=keyword%3Aoled</a></td>
</tr>
<tr>
<td>tft</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Atft">https://platformio.org/lib/search?query=keyword%3Atft</a></td>
</tr>
<tr>
<td>control</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Acontrol">https://platformio.org/lib/search?query=keyword%3Acontrol</a></td>
</tr>
<tr>
<td>device</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Adevice">https://platformio.org/lib/search?query=keyword%3Adevice</a></td>
</tr>
<tr>
<td>glcd</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Aglcd">https://platformio.org/lib/search?query=keyword%3Aglcd</a></td>
</tr>
<tr>
<td>displaycore</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Adisplaycore">https://platformio.org/lib/search?query=keyword%3Adisplaycore</a></td>
</tr>
<tr>
<td>font</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Afont">https://platformio.org/lib/search?query=keyword%3Afont</a></td>
</tr>
<tr>
<td>other</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Aother">https://platformio.org/lib/search?query=keyword%3Aother</a></td>
</tr>
<tr>
<td>i2c</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Ai2c">https://platformio.org/lib/search?query=keyword%3Ai2c</a></td>
</tr>
<tr>
<td>input</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Ainput">https://platformio.org/lib/search?query=keyword%3Ainput</a></td>
</tr>
<tr>
<td>signal</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3ASignal">https://platformio.org/lib/search?query=keyword%3ASignal</a></td>
</tr>
<tr>
<td>sensor</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Asensor">https://platformio.org/lib/search?query=keyword%3Asensor</a></td>
</tr>
<tr>
<td>output</td>
<td><a href="https://platformio.org/lib/search?query=keyword%3Aoutput">https://platformio.org/lib/search?query=keyword%3Aoutput</a></td>
</tr>
<tr>
<td>Name</td>
<td>Url</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>PubSubClient</td>
<td><a href="https://platformio.org/lib/show/89/PubSubClient">https://platformio.org/lib/show/89/PubSubClient</a></td>
</tr>
<tr>
<td>DHT sensor library</td>
<td><a href="https://platformio.org/lib/show/19/DHT%20sensor%20library">https://platformio.org/lib/show/19/DHT%20sensor%20library</a></td>
</tr>
<tr>
<td>ESPAsyncUDP</td>
<td><a href="https://platformio.org/lib/show/359/ESPAsyncUDP">https://platformio.org/lib/show/359/ESPAsyncUDP</a></td>
</tr>
<tr>
<td>NtpClientLib</td>
<td><a href="https://platformio.org/lib/show/727/NtpClientLib">https://platformio.org/lib/show/727/NtpClientLib</a></td>
</tr>
<tr>
<td>Embedis</td>
<td><a href="https://platformio.org/lib/show/408/Embedis">https://platformio.org/lib/show/408/Embedis</a></td>
</tr>
<tr>
<td>Blynk</td>
<td><a href="https://platformio.org/lib/show/415/Blynk">https://platformio.org/lib/show/415/Blynk</a></td>
</tr>
<tr>
<td>SimpleTimer</td>
<td><a href="https://platformio.org/lib/show/419/SimpleTimer">https://platformio.org/lib/show/419/SimpleTimer</a></td>
</tr>
<tr>
<td>Adafruit DHT Unified</td>
<td><a href="https://platformio.org/lib/show/18/Adafruit%20DHT%20Unified">https://platformio.org/lib/show/18/Adafruit%20DHT%20Unified</a></td>
</tr>
</tbody>
</table>

**FEATURED: TODAY**

<table>
<thead>
<tr>
<th>Name</th>
<th>Url</th>
</tr>
</thead>
</table>

**FEATURED: WEEK**

<table>
<thead>
<tr>
<th>Name</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>PubSubClient</td>
<td><a href="https://platformio.org/lib/show/89/PubSubClient">https://platformio.org/lib/show/89/PubSubClient</a></td>
</tr>
</tbody>
</table>

---

1.3. PlatformIO Core (CLI)
**DHT sensor library**  
https://platformio.org/lib/show/19/DHT%20sensor%20library  

**Adafruit Unified Sensor**  
https://platformio.org/lib/show/31/Adafruit%20Unified%20Sensor  

**Blynk**  
https://platformio.org/lib/show/415/Blynk  

**ESPAsyncWebServer**  
https://platformio.org/lib/show/306/ESPAsyncWebServer  

**Adafruit GFX Library**  
https://platformio.org/lib/show/13/Adafruit%20GFX%20Library  

**I2CdevLib-Core**  
https://platformio.org/lib/show/11/I2CdevLib-Core  

**TimeAlarms**  
https://platformio.org/lib/show/68/TimeAlarms  

**PubSubClient**  
https://platformio.org/lib/show/89/PubSubClient  

**Timer**  
https://platformio.org/lib/show/75/Timer  

**esp8266_mdns**  
https://platformio.org/lib/show/1091/esp8266_mdns  

---

**FEATURED: MONTH**

<table>
<thead>
<tr>
<th>Name</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArduinoJson</td>
<td><a href="https://platformio.org/lib/show/64/ArduinoJson">https://platformio.org/lib/show/64/ArduinoJson</a></td>
</tr>
<tr>
<td>DHT sensor library</td>
<td><a href="https://platformio.org/lib/show/19/DHT%20sensor%20library">https://platformio.org/lib/show/19/DHT%20sensor%20library</a></td>
</tr>
<tr>
<td>PubSubClient</td>
<td><a href="https://platformio.org/lib/show/89/PubSubClient">https://platformio.org/lib/show/89/PubSubClient</a></td>
</tr>
<tr>
<td>OneWire</td>
<td><a href="https://platformio.org/lib/show/1/OneWire">https://platformio.org/lib/show/1/OneWire</a></td>
</tr>
<tr>
<td>ESPAsyncTCP</td>
<td><a href="https://platformio.org/lib/show/305/ESPAsyncTCP">https://platformio.org/lib/show/305/ESPAsyncTCP</a></td>
</tr>
<tr>
<td>Time</td>
<td><a href="https://platformio.org/lib/show/44/Time">https://platformio.org/lib/show/44/Time</a></td>
</tr>
<tr>
<td>DallasTemperature</td>
<td><a href="https://platformio.org/lib/show/54/DallasTemperature">https://platformio.org/lib/show/54/DallasTemperature</a></td>
</tr>
<tr>
<td>WifiManager</td>
<td><a href="https://platformio.org/lib/show/567/WifiManager">https://platformio.org/lib/show/567/WifiManager</a></td>
</tr>
</tbody>
</table>

---

**pio lib uninstall**

**Contents**

- **pio lib uninstall**
  - Usage
  - Description
  - Storage Options
  - Options
  - Examples

**Usage**

```
pio lib [STORAGE_OPTIONS] uninstall [LIBRARY...]
```

# uninstall project dependent library  
# (run it from a project root where is located "platformio.ini")

(continues on next page)
pio lib uninstall [LIBRARY...]

# uninstall dependencies for the specific project environment
# (run it from a project root where is located "platformio.ini")
pio lib -e myenv uninstall [OPTIONS] [LIBRARY...]
pio lib -d /path/to/platformio/project -e myenv uninstall [OPTIONS] [LIBRARY...]

# uninstall library from global storage
pio lib --global uninstall [LIBRARY...]
pio lib -g uninstall [LIBRARY...]

# uninstall library from custom storage
pio lib --storage-dir /path/to/dir uninstall [LIBRARY...]
pio lib -d /path/to/dir uninstall [LIBRARY...]

# [LIBRARY...] forms
pio lib [STORAGE_OPTIONS] uninstall <id>
pio lib [STORAGE_OPTIONS] uninstall <id>@<version>
pio lib [STORAGE_OPTIONS] uninstall <id>@<version range>
pio lib [STORAGE_OPTIONS] uninstall <name>
pio lib [STORAGE_OPTIONS] uninstall <name>@<version>
pio lib [STORAGE_OPTIONS] uninstall <name>@<version range>

Description

Uninstall specified library

The version supports Semantic Versioning (<major>.<minor>.<patch>) and can take any of the following forms:

- 1.2.3 - an exact version number. Use only this exact version
- ^1.2.3 - any compatible version (exact version for 1.x.x versions)
- ~1.2.3 - any version with the same major and minor versions, and an equal or greater patch version
- >1.2.3 - any version greater than 1.2.3. >=, <, and <= are also possible
- >0.1.0, !=0.2.0, <0.3.0 - any version greater than 0.1.0, not equal to 0.2.0 and less than 0.3.0

Storage Options

See base options for Library Manager CLI.

Options

--save / --no-save

Remove libraries from the "platformio.ini" (Project Configuration File) dependency list (lib_deps) and save changes. Default value is to save.

-s, --silent

Suppress progress reporting.

-f, --force
Reinstall library if it is already installed.

**Examples**

```shell
> pio lib -g uninstall AsyncMqttClient
Library Storage: /storage/dir/...
Uninstalling AsyncMqttClient @ 0.2.0: [OK]
```

**pio lib update**

**Contents**

- **pio lib update**
  - Usage
  - Description
  - Storage Options
  - Options
  - Examples

**Usage**

```shell
pio lib [STORAGE_OPTIONS] update [OPTIONS]

# update all project libraries
# (run it from a project root where is located "platformio.ini")
pio lib update [OPTIONS]

# update project dependent library
pio lib [STORAGE_OPTIONS] update [OPTIONS] [LIBRARY...]

# update library in global storage
pio lib --global update [OPTIONS] [LIBRARY...]
pio lib -g update [OPTIONS] [LIBRARY...]

# update library in custom storage
pio lib --storage-dir /path/to/dir update [OPTIONS] [LIBRARY...]
pio lib -d /path/to/dir update [OPTIONS] [LIBRARY...]

# [LIBRARY...] forms
pio lib [STORAGE_OPTIONS] update <id>
pio lib [STORAGE_OPTIONS] update <id>@<version>
pio lib [STORAGE_OPTIONS] update <id>@<version range>
pio lib [STORAGE_OPTIONS] update <name>
pio lib [STORAGE_OPTIONS] update <name>@<version>
pio lib [STORAGE_OPTIONS] update <name>@<version range>
```
Description

Check or update installed libraries.

The version supports Semantic Versioning (<major>.<minor>.<patch>) and can take any of the following forms:

- 1.2.3 - an exact version number. Use only this exact version
- ^1.2.3 - any compatible version (exact version for 1.x.x versions)
- ~1.2.3 - any version with the same major and minor versions, and an equal or greater patch version
- >1.2.3 - any version greater than 1.2.3. >=, <, and <= are also possible
- >0.1.0, !=0.2.0, <0.3.0 - any version greater than 0.1.0, not equal to 0.2.0 and less than 0.3.0

Storage Options

See base options for Library Manager CLI.

Options

-c, --only-check
DEPRECATED. Please use --dry-run instead.

--dry-run
Do not update, only check for the new versions

--json-output
Return the output in JSON format

Examples

1. Update all installed libraries in global storage

```
> pio lib -g update
Library Storage: /storage/dir/...
Updating ESP8266_SSD1306 @ 3.2.3: [Up-to-date]
Updating EngduinoMagnetometer @ 3.1.0: [Up-to-date]
Updating IRremote @ 2.2.1: [Up-to-date]
Updating Json @ 5.4.0: [Out-of-date]
LibraryManager: Installing id=64 @ 5.6.4
Downloading [####################################] 100%
Unpacking [####################################] 100%
Json @ 5.6.4 has been successfully installed!
Updating PJON @ 1fb26fd: [Checking]
git version 2.7.4 (Apple Git-66)
Already up-to-date.
Updating TextLCD @ 308d188a2d3a: [Checking]
Mercurial Distributed SCM (version 3.8.4)
(see https://mercurial-scm.org for more information)
```
2. Update specified libraries in global storage

```shell
> pio lib -g update Json 4
Library Storage: /storage/dir/...
Updating Json @ 5.6.4: [Up-to-date]
Updating IRremote @ 2.2.1: [Up-to-date]
```

### pio org

New in version 5.0.

Manage organizations and their owners.

Management of teams and team memberships is done with the `pio team` command.

To print all available commands and options use:

```shell
pio org --help
pio org COMMAND --help
```

### pio org add

New in version 5.0.

#### Contents

- **pio org add**
  - **Usage**
  - **Description**
  - **Examples**
  - **See Also**

#### Usage

```shell
pio org add ORGNAME USERNAME
```

#### Description

Add a user as an owner to an organization.
If you need to create a new organization, please use `pio org create` command.

**Examples**

Add Bob as an owner of “platformio” organization:

```bash
> pio org add platformio bob
The new owner "bob" has been successfully added to the "platformio" organization.
```

**See Also**

- `pio org create`
- `pio org list`
- `pio org remove`

**pio org create**

New in version 5.0.

**Contents**

- `pio org create`
  - Usage
  - Description
    * Options
  - Examples
  - See Also

**Usage**

```bash
pio org create [OPTIONS] ORGNAME
```

**Description**

Create a new organization.

**Options**

--email
An organization e-mail.

--displayname
An organization display name (company name).

**Examples**

```bash
> pio org create platformio --email contact@platformio.org --displayname PlatformIO
The organization "platformio" has been successfully created.
```

**See Also**

- `pio org list`
- `pio org remove`

**pio org destroy**

New in version 5.0.

**Contents**

- `pio org destroy`
  - Usage
  - Description
  - See Also

**Usage**

`pio org destroy ORGNAME`

**Description**

Destroy an organization and its teams.

If you need to remove a user from an organization, please use `pio org remove` command.

**See Also**

- `pio org create`
- `pio org list`
- `pio org update`
pio org list

New in version 5.0.

Contents

- pio org list
  - Usage
  - Description
  - Options
  - Examples

Usage

```
pio org list [OPTIONS] [OWNER]
```

Description

List organizations and their owners.

Options

```
--json-output
```

Return the output in JSON format

Examples

```
> pio org list
...
platformio
--------
Display Name: PlatformIO
Owners: alice, bob
...
```

pio org remove

New in version 5.0.
Usage

`pio org remove ORGNAME USERNAME`

Description

Remove a user from an organization.

If you need to destroy an existing organization, please use `pio org destroy` command.

Examples

Remove Bob from “platformio” organization:

```
> pio org remove platformio bob
The `bob` owner has been successfully removed from the `platformio` organization.
```

See Also

- `pio org add`
- `pio org list`

pio org update

New in version 5.0.
Usage

```
pio org update [OPTIONS] ORGNAME
```

Description

Rename the organization or update the existing details.

Options

```
--orgname
```
New organization “orgname”. Must contain only alphanumeric characters or single hyphens, cannot begin or end with a hyphen, and must not be longer than 38 characters.

```
--email
```
An organization e-mail.

```
--displayname
```
An organization name (company name).

Examples

```
> pio org update platformio --email contact@platformio.org --displayname PlatformIO
The organization "platformio" has been successfully updated.
```

See Also

- `pio org list`
- `pio org remove`

pio package

Manage packages in the registry.

To print all available commands and options use:

```
pio package --help
pio package COMMAND --help
```

pio package pack

New in version 5.0.
Contents

- pio package pack
  - Usage
  - Description
    - Options
  - See Also

Usage

pio package pack [<source directory, tar.gz or zip>] [OPTIONS]

Description

Create a tarball from a package (library, Development Platforms, or tool).
If no arguments are supplied, then platformio packs the current package folder.
A source of a package must contain a manifest (should be located in a root of a package) depending on a package type:

Library A library.json is a manifest file of a library package.

Platform A platform.json is a manifest file of a development platform package. See Manifest File platform.json.

Tool A package.json is a manifest file of a tool package. It is intended for toolchains, flashers, debugging servers, simulators, and other tools.

See a manifest validation scheme.

Options

-o, --output

Specify a destination path (folder or a full path to file) where to store a tarball. The default is to create a tarball in the current working directory.

See Also

- library.json
- pio package publish

pio package publish

New in version 5.0.
Usage

```shell
pio package publish [<source directory, tar.gz or zip>] [OPTIONS]
```

Description

Publish a package to the registry so that it can be installed by name. All files in the package directory are included if `export` field is not declared in a package manifest (for example, see `export`). The next items are automatically excluded.

Please check which files will be included in the final package using the `pio package pack` command.

Once a package is published with a given name and version, that specific name and version combination can never be used again, even if it is removed with the `pio package unpublish` command.

If no arguments are supplied, then PlatformIO packs the current package folder.

To list published packages, please use `pio access list` command.

Options

```
--owner
PlatformIO Account username (can be organization username). The default is set to a username of the authorized PlatformIO Account.

--released-at
Custom release date and time in the next format (UTC): 2014-06-13 17:08:52

--private
```

Note: The permission to set custom release dates and times is only available to Super Admins.

Restrict access to a package (will not be available publicly). The default is to publish a package publicly.

```
--no-notify
```

Do not notify by email when package is processed. The default behavior is to notify.
See Also

- `library.json`
- `pio package pack`
- `pio package unpublish`

**pio package unpublish**

New in version 5.0.

### Contents

- `pio package unpublish`
  - **Usage**
  - **Description**
    - * Options
  - **See Also**

### Usage

```sh
pio package unpublish [<organization>/]<pkgname>[@<version>] [OPTIONS]
```

### Description

This removes a package version from the registry, deleting its entry and removing the tarball.

If no version is specified, or if all versions are removed then the root package entry is removed from the registry entirely.

You can only remove a package version within 72 hours since the published date.

### Options

---`type`

Set the type of a package to unpublish (if you have packages with the same name). Possible values are `library`, `platform`, or `tool`. The default is set to `library`.

---`undo`

Undo a remove, putting a version back into the registry.

### See Also

- `library.json`
- `pio package pack`
• *pio package publish*

**Platform Manager CLI**

To print all available commands and options use:

```
pio platform --help
pio platform COMMAND --help
```

**pio platform frameworks**

**Contents**

- *pio platform frameworks*
  - Usage
  - Description
  - Options
  - Examples

**Usage**

```
pio platform frameworks QUERY [OPTIONS]
```

**Description**

List supported *Frameworks* (SDKs, etc).

**Options**

--json-output

Return the output in *JSON* format

**Examples**

Print all supported frameworks, SDKs, etc.

```
> pio platform frameworks
arduino ~ Arduino
===============
Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
```

(continues on next page)
artik-sdk ~ ARTIK SDK
======================

ARTIK SDK is a C/C++ SDK targeting Samsung ARTIK platforms. It exposes a set of APIs to ease up development of applications. These APIs cover hardware buses such as GPIO, SPI, I2C, UART, connectivity links like Wi-Fi, Bluetooth, Zigbee, and network protocols such as HTTP, Websockets, MQTT, and others.

Home: https://platformio.org/frameworks/artik-sdk

cmsis ~ CMSIS
-------------

The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.

Home: https://platformio.org/frameworks/cmsis

espidf ~ ESP-IDF
===============


Home: https://platformio.org/frameworks/espidf

libopencm3 ~ libOpenCM3
=======================

The libOpenCM3 framework aims to create a free/libre/open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC 11xx, 13xx, 15xx, 17xx parts, Atmel SAM3, Energy Micro EFM32 and others.

Home: https://platformio.org/frameworks/libopencm3

mbed ~ mbed
-----------

The mbed framework The mbed SDK has been designed to provide enough hardware abstraction to be intuitive and concise, yet powerful enough to build complex projects. It is built on the low-level ARM CMSIS APIs, allowing you to code down to the metal if needed. In addition to RTOS, USB and Networking libraries, a cookbook of hundreds of reusable peripheral and module libraries have been built on top of the SDK by the mbed Developer Community.

Home: https://platformio.org/frameworks/mbed

pumbaa ~ Pumbaa
-------------

Pumbaa is Python on top of Simba. The implementation is a port of MicroPython, designed for embedded devices with limited amount of RAM and code memory.

Home: https://platformio.org/frameworks/pumbaa

simba ~ Simba
--------------

(continues on next page)
Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Home: https://platformio.org/frameworks/simba

spl ~ SPL

The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 Cortex-M3 family. The idea is to save the user (the new user, in particular) having to deal directly with the registers.

Home: https://platformio.org/frameworks/spl

wiringpi ~ WiringPi

WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry Pi. It's designed to be familiar to people who have used the Arduino "wiring" system.

Home: https://platformio.org/frameworks/wiringpi

pio platform install

Contents

- pio platform install
  - Usage
  - Options
  - Description
  - Version control
    - Git
    - Mercurial
    - Subversion
  - Examples

Usage

pio platform install [OPTIONS] [PLATFORM...]

# [PLATFORM...] forms
pio platform install <name>
pio platform install <name>@<version>
pio platform install <name>@<version range>
pio platform install <ownername>@<version>
pio platform install <ownername>@<version range>

(continues on next page)
pio platform install <zip or tarball url>
pio platform install file://<zip or tarball file>
pio platform install file://<folder>
pio platform install <repository>
pio platform install <name=repository> (name it should have locally)
pio platform install <repository#tag> ("tag" can be commit, branch or tag)

Options

--with-package
Install specified package (or alias)
--without-package
Do not install specified package (or alias)
--skip-default
Skip default packages
--with-all-packages
Install all declared packages in platform.json
-f, --force
Reinstall/redownload development platform and its packages if they exist

Description

Install Development Platforms and dependent packages.
The version supports Semantic Versioning (<major>.<minor>.<patch>) and can take any of the following forms:

- ^1.2.3 - any compatible version (new functionality in a backwards compatible manner and patches are allowed, 1.x.x). **RECOMMENDED**
- ~1.2.3 - any version with the same major and minor versions, and an equal or greater patch version
- >1.2.3 - any version greater than 1.2.3. >=, <, and <= are also possible
- >0.1.0, !=0.2.0, <0.3.0 - any version greater than 0.1.0, not equal to 0.2.0 and less than 0.3.0
- 1.2.3 - an exact version number. Use only this exact version.

Also, PlatformIO supports installing from local directory or archive. Need to use file:// prefix before local path. Also, directory or archive should contain platform.json manifest.

- file:///local/path/to/the/platform/dir
- file:///local/path/to/the/platform.zip
- file:///local/path/to/the/platform.tar.gz
Version control

PlatformIO supports installing from Git, Mercurial and Subversion, and detects the type of VCS using url prefixes: “git+”, “hg+”, or “svn+”.

**Note:** PlatformIO requires a working VCS command on your path: `git`, `hg` or `svn`.

**Git**

The supported schemes are: `git`, `git+https` and `git+ssh`. Here are the supported forms:

- `https://github.com/platformio/platform-NAME.git`
- `git+git://git.server.org/my-platform`
- `git+https://git.server.org/my-platform`
- `git+ssh://git.server.org/my-platform`
- `git+ssh://user@git.server.org/my-platform`
- `[user@]host.xz:path/to/repo.git`

Passing branch names, a commit hash or a tag name is possible like so:

- `https://github.com/platformio/platform-name.git#master`
- `git+git://git.server.org/my-platform#master`
- `git+https://git.server.org/my-platform#v1.0`
- `git+ssh://git.server.org/my-platform#7846d8ad52f983f2f2887bdc0f073fe9755a806d`

**Mercurial**

The supported schemes are: `hg+http`, `hg+https` and `hg+ssh`. Here are the supported forms:

- `hg+http://hg.server.org/my-platform`
- `hg+https://hg.server.org/my-platform`
- `hg+ssh://hg.server.org/my-platform`

Passing branch names, a commit hash or a tag name is possible like so:

- `hg+http://hg.server.org/my-platform#master`
- `hg+https://hg.server.org/my-platform#v1.0`
- `hg+ssh://hg.server.org/my-platform#4cfe2fa00668`

**Subversion**

The supported schemes are: `svn`, `svn+svn`, `svn+http`, `svn+https` and `svn+ssh`. Here are the supported forms:

- `svn+svn://svn.server.org/my-platform`
- `svn+https://svn.server.org/my-platform`
Examples

1. Install Atmel AVR with default packages

```
> pio platform install atmelavr

PlatformManager: Installing atmelavr
Downloading...
Unpacking [####################################] 100%
atmelavr @ 0.0.0 has been successfully installed!
PackageManager: Installing tool-scons @ >=2.3.0,<2.6.0
Downloading [####################################] 100%
Unpacking [####################################] 100%
tool-scons @ 2.4.1 has been successfully installed!
PackageManager: Installing toolchain-atmelavr @ ~1.40801.0
Downloading [####################################] 100%
Unpacking [####################################] 100%
toolchain-atmelavr @ 1.40801.0 has been successfully installed!
The platform 'atmelavr' has been successfully installed!
The rest of packages will be installed automatically depending on your build environment.
```

2. Install Atmel AVR with uploader utility only and skip default packages

```
> pio platform install atmelavr --skip-default-package --with-package=uploader

PlatformManager: Installing atmelavr
Downloading [####################################] 100%
Unpacking [####################################] 100%
atmelavr @ 0.0.0 has been successfully installed!
PackageManager: Installing tool-micronucleus @ ~1.200.0
Downloading [####################################] 100%
Unpacking [####################################] 100%
tool-micronucleus @ 1.200.0 has been successfully installed!
PackageManager: Installing tool-avrdude @ ~1.60001.0
Downloading [####################################] 100%
Unpacking [####################################] 100%
tool-avrdude @ 1.60001.1 has been successfully installed!
The platform 'atmelavr' has been successfully installed!
The rest of packages will be installed automatically depending on your build environment.
```

3. Install the latest development Atmel AVR from Git repository

```
> pio platform install https://github.com/platformio/platform-atmelavr.git

PlatformManager: Installing platform-atmelavr
git version 2.7.4 (Apple Git-66)
Cloning into '/Volumes/MEDIA/tmp/pio3_test_projects/arduino-digihead-master/home_dir/…
remote: Counting objects: 176, done.
```
remote: Compressing objects: 100% (55/55), done.
remote: Total 176 (delta 114), reused 164 (delta 109), pack-reused 0
Receiving objects: 100% (176/176), 38.86 KiB | 0 bytes/s, done.
Resolving deltas: 100% (114/114), done.
Checking connectivity... done.
Submodule 'examples/arduino-external-libs/lib/OneWire' (https://github.com/ —PaulStoffregen/OneWire.git) registered for path 'examples/arduino-external-libs/lib/ —OneWire'
Cloning into 'examples/arduino-external-libs/lib/OneWire'...
remote: Counting objects: 91, done.
remote: Total 91 (delta 0), reused 0 (delta 0), pack-reused 91
Unpacking objects: 100% (91/91), done.
Checking connectivity... done.
Submodule path 'examples/arduino-external-libs/lib/OneWire': checked out
—'57c18c6de80c13429275f70875c7c341f1719201'
atmelavr @ 0.0.0 has been successfully installed!
PackageManager: Installing tool-scons @ >=2.3.0,<2.6.0
Downloading [####################################] 100%
Unpacking [####################################] 100%
tool-scons @ 2.4.1 has been successfully installed!
PackageManager: Installing toolchain-atmelavr @ ~1.40801.0
Downloading [####################################] 100%
Unpacking [####################################] 100%
toolchain-atmelavr @ 1.40801.0 has been successfully installed!
The platform 'https://github.com/platformio/platform-atmelavr.git' has been—successfully installed!
The rest of packages will be installed automatically depending on your build—environment.

pio platform list

Contents

• pio platform list
  — Usage
  — Description
    * Options
  — Examples

Usage

pio platform list [OPTIONS]

Description

List installed Development Platforms
Options

|--json-output
Return the output in JSON format.

Examples

> pio platform list
atmelavr ~ Atmel AVR
====================
Atmel AVR 8- and 32-bit MCUs deliver a unique combination of performance, power, efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones—they are based on the industry’s most code-efficient architecture, for C and assembly programming.

Home: https://platformio.org/platforms/atmelavr
Packages: toolchain-atmelavr, framework-simba
Version: 0.0.0

atmelsam ~ Atmel SAM
=====================
Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3, and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

Home: https://platformio.org/platforms/atmelsam
Packages: framework-arduinosam, framework-mbed, framework-simba, toolchain--gccarmnoneeabi, tool-bossac
Version: 0.0.0

espressif8266 ~ Espressif 8266
=================================
Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Home: https://platformio.org/platforms/espressif8266
Packages: framework-simba, tool-esptool, framework-arduinoespressif8266, sdk-esp8266, --toolchain-xtensa
Version: 0.0.0
...

pio platform search

Contents

- pio platform search
  - Usage
  - Description
**Options**

- **Examples**

**Usage**

```
pio platform search QUERY [OPTIONS]
```

**Description**

Search for development Development Platforms

**Options**

`--json-output`

Return the output in JSON format

**Examples**

1. Print all available development platforms

```
> pio platform search

atmelavr ~ Atmel AVR
====================
Atmel AVR 8- and 32-bit MCUs deliver a unique combination of performance, power, efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industrys most code-efficient architecture for C and assembly programming.

Home: https://platformio.org/platforms/atmelavr
Packages: toolchain-atmelavr, framework-arduinoavr, framework-simba, tool-avrdude,

atmelsam ~ Atmel SAM
====================
Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

Home: https://platformio.org/platforms/atmelsam
Packages: toolchain-gccarmnoneeabi, framework-arduinosa, framework-simba, tool-

espressif32 ~ Espressif 32
============================
Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
```
2. Search for TI development platforms

> pio platform search texas

timsp430 ~ TI MSP430

MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.

Home: https://platformio.org/platforms/timsp430

Packages: toolchain-timsp430, tool-mspdebug, framework-energiamsp430, framework-arduinomsp430

titiva ~ TI TIVA

Texas Instruments TM4C12x MCUs offer the industrys most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.

Home: https://platformio.org/platforms/titiva

Packages: ldscripts, framework-libopencm3, toolchain-gccarmnoneeabi, tool-lm4flash, framework-energiativa

> pio platform search framework-mbed

atmelsam ~ Atmel SAM

Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

Home: https://platformio.org/platforms/atmelsam

(continues on next page)
Packages: toolchain-gccarmnoneeabi, framework-arduinosam, framework-simba, tool-
→openocd, framework-mbed, ldscripts, tool-bossac

freescalekinetis ~ Freescale Kinetics
====================================
Freescale Kinetics Microcontrollers is family of multiple hardware- and software-
→compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetics MCUs,
→offer exceptional low-power performance, scalability and feature integration.

Home: https://platformio.org/platforms/freescalekinetis
Packages: framework-mbed, toolchain-gccarmnoneeabi

nordicnrf51 ~ Nordic nRF51
==========================
The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-
→chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices,
→support a range of protocol stacks including Bluetooth Smart (previously called,
→Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

Home: https://platformio.org/platforms/nordicnrf51
Packages: framework-mbed, tool-rfdloader, toolchain-gccarmnoneeabi, framework-
→arduinonordicnrf51

nxplpc ~ NXP LPC
==============
The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP.
→Semiconductors. The LPC chips are grouped into related series that are based around
→the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+
→or Cortex-M0. Internally, each microcontroller consists of the processor core,
→static RAM memory, flash memory, debugging interface, and various peripherals.

Home: https://platformio.org/platforms/nxplpc
Packages: framework-mbed, toolchain-gccarmnoneeabi

siliconlabsefm32 ~ Silicon Labs EFM32
=====================================
Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that,
→offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to
→48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features,
→innovative low energy techniques, short wake-up time from energy saving modes and a
→wide selection of peripherals, making it ideal for battery operated applications,
→and other systems requiring high performance and low-energy consumption.

Home: https://platformio.org/platforms/siliconlabsefm32
Packages: framework-mbed, toolchain-gccarmnoneeabi

ststm32 ~ ST STM32
==============
The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed,
→to offer new degrees of freedom to MCU users. It offers a 32-bit product range that,
→combines very high performance, real-time capabilities, digital signal processing,
→and low-power, low-voltage operation, while maintaining full integration and ease
→of development.

Home: https://platformio.org/platforms/ststm32
Packages: framework-libopencm3, toolchain-gccarmnoneeabi, tool-stlink, framework-spl,
→framework-cmsis, framework-mbed, ldscripts

(continues on next page)
teensy ~ Teensy
===============
Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

Home: https://platformio.org/platforms/teensy
Packages: framework-arduinoteensy, tool-teensy, toolchain-gccarmnoneeabi, framework-mbed, toolchain-atmelavr, ldscripts

pio platform show

Contents

- pio platform show
  - Usage
  - Description
  - Examples

Usage

pio platform show PLATFORM

Description

Show details about Development Platforms

Examples

> pio platform show atmelavr

atmelavr ~ Atmel AVR
======================
Atmel AVR 8- and 32-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.

Version: 1.2.1
Home: https://platformio.org/platforms/atmelavr
License: Apache-2.0
Frameworks: simba, arduino

(continues on next page)
Package toolchain-atmelavr
--------------------------
Type: toolchain
Requirements: ~1.40902.0
Installed: Yes
Description: avr-gcc
Version: 1.40902.0 (4.9.2)

Package framework-arduinoavr
-----------------------------
Type: framework
Requirements: ~1.10612.1
Installed: Yes
Url: https://www.arduino.cc/en/Main/Software
Version: 1.10612.1 (1.6.12)
Description: Arduino Wiring-based Framework (AVR Core, 1.6)

Package framework-simba
------------------------
Type: framework
Requirements: >=7.0.0
Installed: Yes
Url: https://github.com/eerimoq/simba
Version: 11.0.0
Description: Simba Embedded Programming Platform

Package tool-avrdude
---------------------
Type: uploader
Requirements: ~1.60300.0
Installed: Yes
Description: AVRDUDE
Url: http://www.nongnu.org/avrdude/
Version: 1.60300.0 (6.3.0)

Package tool-micronucleus
-------------------------
Type: uploader
Requirements: ~1.200.0
Installed: No (optional)

pio platform uninstall

Contents

- pio platform uninstall
  - Usage
  - Description
  - Examples
Usage

```
pio platform uninstall [PLATFORM...]

# uninstall specific platform version using Semantic Versioning
pio platform uninstall PLATFORM@X.Y.Z
```

Description

Uninstall specified Development Platforms

Examples

```
> pio platform uninstall atmelavr
Uninstalling platform atmelavr @ 0.0.0: [OK]
Uninstalling package tool-scons @ 2.4.1: [OK]
Uninstalling package toolchain-atmelavr @ 1.40801.0: [OK]
The platform 'atmelavr' has been successfully uninstalled!
```

pio platform update

Contents

- pio platform update
  - Usage
  - Description
  - Options
  - Examples

Usage

```
pio platform update [OPTIONS] [PLATFORM...]

# update specific platform version using Semantic Versioning
pio platform update PLATFORM@X.Y.Z
```

Description

Check or update installed Development Platforms

Options

- `p`, `--only-packages`
Update only the platform related packages. Do not update development platform build scripts, board configs and etc.

`-c, --only-check`

DEPRECATED. Please use `--dry-run` instead.

`--dry-run`

Do not update, only check for the new versions

`--json-output`

Return the output in JSON format

**Examples**

```
> pio platform update

Platform atmelavr
--------
Updating atmelavr @ 0.0.0: [Up-to-date]
Updating framework-arduinavr @ 1.10608.1: [Up-to-date]
Updating tool-avrdude @ 1.60001.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform espressif8266
--------
Updating espresif @ 0.0.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-xtensa @ 1.40802.0: [Up-to-date]
Updating tool-esptool @ 1.409.0: [Up-to-date]
Updating tool-mkspiffs @ 1.102.0: [Up-to-date]
Updating framework-arduinoespressif8266 @ 1.20300.0: [Up-to-date]
Updating sdk-esp8266 @ 1.10502.0: [Up-to-date]

Platform teensy
--------
Updating teensy @ 0.0.0: [Up-to-date]
Updating framework-arduinoteensy @ 1.128.0: [Up-to-date]
Updating tool-teensy @ 1.1.0: [Up-to-date]
Updating framework-mbed @ 1.121.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
```

**Project Manager CLI**

To print all available commands and options use:

```
pio project --help
pio project COMMAND --help
```
pio project config

Contents

• pio project config
  – Usage
  – Description
    • Options
  – Examples

Usage

pio project config [OPTIONS]

Description

Show project computed configuration based on “platformio.ini” (Project Configuration File). The extra configuration files and dynamic variables will be expanded.

This command is useful for developers to check how PlatformIO computes configuration from “platformio.ini” (Project Configuration File).

Options

-d, --project-dir

Specify the path to project directory. By default, --project-dir is equal to current working directory (CWD).

--json-output

Return the output in JSON format.

Examples

> pio project config
Computed project configuration for Tasmota Project

platformio
---------
src_dir = tasmota
build_dir = .pioenvs
build_cache_dir = .cache
extra_configs = platformio_tasmota_env.ini
                  platformio_override.ini
default_envs = tasmota
common

(continues on next page)
### pio project data

New in version 5.0.

#### Contents

- pio project data
  - Usage
  - Description
  - Options

#### Usage

```
pio project data [OPTIONS]
```
Description

Dump data intended for IDE extensions/plugins:

- Toolchain type and location
- Compiler flags
- Defines/Macros
- CPP Preprocessor includes/paths
- Program path
- SVD path if available for Development Platforms
- Targets by Development Platforms (see `pio run --list-targets` for details)
- Extra information.

Options

-d, --project-dir
Specify the path to project directory. By default, `--project-dir` is equal to current working directory (CWD).

-e, --environment
Dump specified environments. Multiple environments are allowed.

--json-output
Return the output in JSON format.

pio project init

Usage

```
pio project init [OPTIONS]
```
Description

Initialize a new PlatformIO based project or update existing with new data.

This command will create:

- "platformio.ini" (Project Configuration File)
- \texttt{include\_dir}, put project header files here
- \texttt{src\_dir}, put project source files here (*.h, *.c, *.cpp, *.S, *.ino, etc.)
- \texttt{lib\_dir}, put project specific (private) libraries here. See also \textit{Library Dependency Finder (LDF)}
- \texttt{test\_dir}, put project tests here. More details \textit{Unit Testing}
- Miscellaneous files for VCS and \textit{Continuous Integration} support.

Options

- \texttt{-d, --project-dir}
  A path to a directory where \texttt{PlatformIO} will initialize new project.

- \texttt{-b, --board}
  If you specify board ID (you can pass multiple \texttt{-b} options), then \texttt{PlatformIO} will automatically generate environment for "platformio.ini" (Project Configuration File) and pre-fill these data:
  - platform
  - framework
  - board
  The full list with pre-configured boards is available here \textit{Development Platforms}.

- \texttt{--ide}
  Initialize PlatformIO project for the specified IDE which can be imported later via "Import Project" functionality.
  A list with supported IDE is available within \texttt{pio project init --help} command. Also, please take a look at \textit{Cloud & Desktop IDE} page.

- \texttt{-e, --environment}
  New in version 5.0.
  Update project using existing environment.

- \texttt{-O, --project-option}
  Initialize project with additional options from "platformio.ini" (Project Configuration File). For example, \texttt{pio project init --project-option="lib\_deps=ArduinoJSON"}. Multiple options are allowed.

- \texttt{--env-prefix}
  An environment prefix which will be used with pair in board ID. For example, the default environment name for Teensy 3.1/3.2 board will be [env:teensy31].

- \texttt{-s, --silent}
  Suppress progress reporting
Examples

1. Initialize new project in a current working directory

```bash
> pio project init
```

The current working directory *** will be used for the new project. You can specify another project directory via `pio project init -d %PATH_TO_THE_PROJECT_DIR%` command.

The next files/directories will be created in ***
- platformio.ini - Project Configuration File. |-> PLEASE EDIT ME <-|
- src - Put your source files here
- lib - Put here project specific (private) libraries

Project has been successfully initialized!

Useful commands:
- `pio run` - process/build project from the current directory
- `pio run --target upload` or `pio run -t upload` - upload firmware to embedded board
- `pio run --target clean` - clean project (remove compiled files)

2. Initialize new project in a specified directory

```bash
> pio project init -d %PATH_TO_DIR%
```

The next files/directories will be created in ***
- platformio.ini - Project Configuration File. |-> PLEASE EDIT ME <-|

3. Initialize project for Arduino Uno

```bash
> pio project init --board uno
```

4. Initialize project for Teensy 3.1 board with custom Mbed

```bash
> pio project init --board teensy31 --project-option "framework=mbed"
```

5. Initialize a new project passing working environment name and its options

```bash
> pio project init --environment native --project-option="platform=native" --project-option="build_flags=-DRELEASE=1"
```

The current working directory *** will be used for the new project.
PlatformIO Remote CLI

Helper command for Remote Development.

To print all available commands and options use:

```
pio remote --help
pio remote COMMAND --help

# run command on the specified PlatformIO Remote Agents
pio remote --agent NAME_1 --agent NAME_N COMMAND
```

PlatformIO Remote Agent

Start PlatformIO Remote Agent on a host machine and work remotely with your devices WITHOUT extra software, services, SSH, VPN, tunneling or opening incoming network ports.

Remote Development supports wired and wireless devices. Wired devices should be connected physically to host machine where PlatformIO Remote Agent is started, where wireless devices should be visible for PlatformIO Remote Agent to provide network operations Over-The-Air (OTA).

Contents

- PlatformIO Remote Agent
  - pio remote agent list
    - Usage
    - Description
    - Example
  - pio remote agent start
    - Usage
    - Description
    - Options

pio remote agent list

Usage

```
pio remote agent list
```

Description

List active PlatformIO Remote Agent s started using own PlatformIO Account or shared with you by other PlatformIO developers.
Example

```bash
> pio remote agent list

innomac.local
-------------
ID: 98853d930....788d77375e7
Started: 2016-10-26 16:32:56
```

`pio remote agent start`

Usage

`pio remote agent start [OPTIONS]`

Description

Start *PlatformIO Remote Agent* and work remotely with your devices from anywhere in the world. This command can be run as daemon or added to autostart list of your OS.

Options

`-n, --name`
Agent name/alias. By default, machine’s `hostname` will be used. You can use this name later for `pio remote device` and `pio remote run` commands. Good names are home, office, lab or etc.

`-s, --share`
Share your agent/devices with other PlatformIO developers who have *PlatformIO Account*: friends, co-workers, team, etc.

The valid value for `--share` option is email address that was used for `pio account register` command.

`-d, --working-dir`
A working directory where *PlatformIO Remote Agent* stores projects data for incremental synchronization and embedded programs for PlatformIO Process Supervisor.

`pio remote device`

Remote Device: monitor remote device or list existing.

Contents

- `pio remote device`
  - `pio remote device list`
pio remote device list

Usage

pio remote device list [OPTIONS]

# List devices from the specified agents. Multiple agents are allowed.
pio remote --agent NAME device list [OPTIONS]

Description

List Serial Ports on remote machines where PlatformIO Remote Agent is started.

You can list devices from the specified remote machines using --agent NAME option between “remote” & “device” sub-commands. For example, you have run pio remote agent start --name option with “home” and “office” options:

• pio remote agent start --name home
• pio remote agent start --name office

Now, to list devices from office machine please use pio remote --agent office device list.

Multiple agents are allowed (pio remote --agent lab1 --agent lab3 device ...).

Options

--json-output
Return the output in JSON format

Example

> pio remote device list
Agent innomac.local

(continues on next page)
pio remote device monitor

Remote Serial Port Monitor

Usage

```bash
pio remote device monitor [OPTIONS]
```

# Connect to a specified agent
```bash
pio remote --agent NAME device monitor [OPTIONS]
pio remote -a NAME device monitor [OPTIONS]
```

Description

Connect to Serial Port of remote device and receive or send data in real time. *PlatformIO Remote Agent* should be started before on a remote machine.

To control *monitor* please use these “hot keys”:

- Ctrl+C Quit
- Ctrl+T Menu
- Ctrl+T followed by Ctrl+H Help
Options

-p, --port
Port, a number or a device name

-b, --baud
Set baud rate, default 9600

--parity
Set parity (None, Even, Odd, Space, Mark), one of [N, E, O, S, M], default N

--rtscts
Enable RTS/CTS flow control, default Off

--xonxoff
Enable software flow control, default Off

--rts
Set initial RTS line state, default 0

--dtr
Set initial DTR line state, default 0

--echo
Enable local echo, default Off

--encoding
Set the encoding for the serial port (e.g. hexlify, Latin1, UTF-8), default UTF-8.

-f, --filter
Add text transformation. Available filters:
  • colorize Apply different colors for received and echo
  • debug Print what is sent and received
  • default Remove typical terminal control codes from input
  • direct Do-nothing: forward all data unchanged
  • nocontrol Remove all control codes, incl. CR+LF
  • printable Show decimal code for all non-ASCII characters and replace most control codes

--eol
End of line mode (CR, LF or CRLF), default CRLF

--raw
Do not apply any encodings/transformation

--exit-char
ASCII code of special character that is used to exit the application, default 3 (DEC, Ctrl+C).
For example, to use Ctrl+] run pio remote device monitor --exit-char 29.

--menu-char
ASCII code of special character that is used to control miniterm (menu), default 20 (DEC)
---quiet
Diagnostics: suppress non-error messages, default Off

-d, --project-dir
Specify the path to project directory. By default, --project-dir is equal to current working directory (CWD).

-e, --environment
Process specified environments.

You can also specify which environments should be processed by default using default_envs option from “platformio.ini” (Project Configuration File).

Examples

1. Show available options for monitor

> pio remote device monitor --help

Usage: pio remote device monitor [OPTIONS]

Options:
-p, --port TEXT  Port, a number or a device name
-b, --baud INTEGER  Set baud rate, default=9600
--parity [N|E|O|S|M]  Set parity, default=N
--rtscts  Enable RTS/CTS flow control, default=Off
--xonxoff  Enable software flow control, default=Off
--rts [0|1]  Set initial RTS line state, default=0
--dtr [0|1]  Set initial DTR line state, default=0
--echo  Enable local echo, default=Off
--encoding TEXT  Set the encoding for the serial port (e.g. hexlify, Latin1, UTF-8), default: UTF-8
-f, --filter TEXT  Add text transformation
--eol [CR|LF|CRLF]  End of line mode, default=CRLF
--raw  Do not apply any encodings/transformations
--exit-char INTEGER  ASCII code of special character that is used to exit the application, default=29 (DEC)
--menu-char INTEGER  ASCII code of special character that is used to control miniterm (menu), default=20 (DEC)
--quiet  Diagnostics: suppress non-error messages, default=Off
-h, --help  Show this message and exit.

2. Communicate with serial device and print help inside terminal

> pio remote device monitor

--- Available ports:
--- /dev/cu.Bluetooth-Incoming-Port n/a
--- /dev/cu.Bluetooth-Modem n/a
--- /dev/cu.SLAB_USBtoUART CP2102 USB to UART Bridge Controller
--- /dev/obd2ecu-SPPDev n/a
Enter port name:/dev/cu.SLAB_USBtoUART
--- Miniterm on /dev/cu.SLAB_USBtoUART: 9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Hello PlatformIO!
---
--- Ctrl+] Exit program

(continues on next page)
pio remote run

Remote Firmware Updates

Contents

• pio remote run
  – Usage
  – Description
  – Options
  – Example

Usage

pio remote run [OPTIONS]

# process environments using specified PlatformIO Remote Agent
pio remote --agent NAME run [OPTIONS]

Description

Process remotely environments which are defined in “platformio.ini” (Project Configuration File) file. By default, Remote Development builds project on a host machine and deploy final firmware (program) to a remote device (embedded board).
If you need to process project on a remote machine, please use `pio remote run --force-remote` option. In this case, `Remote Development` will automatically synchronize your project with remote machine, install required toolchains, frameworks, SDKs, etc., and process project.

**Options**

- **-e, --environment**
  Process specified environments.
  You can also specify which environments should be processed by default using `default_envs` option from “`platformio.ini`” *(Project Configuration File)*.

- **-t, --target**
  Process specified targets. See `pio run --list-targets` documentation for available targets.

- **--upload-port**
  Custom upload port of embedded board. To print all available ports use `pio remote device` command.
  If upload port is not specified, PlatformIO will try to detect it automatically.

- **-d, --project-dir**
  Specify the path to project directory. By default, `--project-dir` is equal to current working directory *(CWD)*.

- **-v, --verbose**
  Shows detailed information when processing environments.
  This option can also be set globally using `force_verbose` setting or by environment variable `PLATFORMIO_SETTING_FORCE_VERBOSE`.

- **--disable-auto-clean**
  Disable auto-clean of `build_dir` when “`platformio.ini`” *(Project Configuration File)* or `src_dir` (project structure) have been modified.

- **-r, --force-remote**
  By default, `Remote Development` builds project on a host machine and deploy final firmware (program) to remote device (embedded board).
  If you need to process project on remote machine, please use `pio remote run --force-remote` option. In this case, `Remote Development` will automatically synchronize your project with remote machine, install required toolchains, frameworks, SDKs, etc., and process project.

**Example**

```
> pio remote run --environment uno --target upload

Building project locally
[Wed Oct 26 16:35:09 2016] Processing uno (platform: atmelavr, board: uno, framework: ...
  --arduino)
-------------------------------------------------------------------------------
Verbose mode can be enabled via `-v, --verbose` option
Collected 25 compatible libraries
Looking for dependencies...
Project does not have dependencies
```

(continues on next page)
Compiling .pio/build/uno/src/main.o
Archiving .pio/build/uno/libFrameworkArduinoVariant.a
Compiling .pio/build/uno/FrameworkArduino/CDC.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial10.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial11.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial12.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial13.o
Compiling .pio/build/uno/FrameworkArduino/IPAddress.o
Compiling .pio/build/uno/FrameworkArduino/PluggableUSB.o
Compiling .pio/build/uno/FrameworkArduino/Print.o
Compiling .pio/build/uno/FrameworkArduino/Stream.o
Compiling .pio/build/uno/FrameworkArduino/Tone.o
Compiling .pio/build/uno/FrameworkArduino/USBCore.o
Compiling .pio/build/uno/FrameworkArduino/WInterrupts.o
Compiling .pio/build/uno/FrameworkArduino/WMath.o
Compiling .pio/build/uno/FrameworkArduino/WString.o
Compiling .pio/build/uno/FrameworkArduino/_wiring_pulse.o
Compiling .pio/build/uno/FrameworkArduino/abi.o
Compiling .pio/build/uno/FrameworkArduino/hooks.o
Compiling .pio/build/uno/FrameworkArduino/main.o
Compiling .pio/build/uno/FrameworkArduino/new.o
Compiling .pio/build/uno/FrameworkArduino/wiring.o
Compiling .pio/build/uno/FrameworkArduino/wiring_analog.o
Compiling .pio/build/uno/FrameworkArduino/wiring_digital.o
Compiling .pio/build/uno/FrameworkArduino/wiring_pulse.o
Compiling .pio/build/uno/FrameworkArduino/wiring_shift.o
Archiving .pio/build/uno/libFrameworkArduino.a
Indexing .pio/build/uno/libFrameworkArduino.a
Linking .pio/build/uno/firmware.elf
Checking program size
Building .pio/build/uno/firmware.hex
text data bss dec hex filename
2574 48 168 2790 ae6 .pio/build/uno/firmware.elf
=======================================================================
[SUCCESS] Took 10.01 seconds
=======================================================================

Environment nodemcuV2 [SKIP]
Environment uno_pic32 [SKIP]
Environment teensy31 [SKIP]
Environment uno [SUCCESS]
=======================================================================
[SUCCESS] Took 10.01 seconds
=======================================================================
Uploading firmware remotely

--------
Verbese mode can be enabled via `-v, --verbose` option
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA1431
Uploading .pio/build/uno/firmware.hex
avrdu: AVR device initialized and ready to accept instructions
Reading | #################################################################### | 100% 0.00s
avrdu: Device signature = 0x1e950f
avrdu: reading input file ".pio/build/uno/firmware.hex"
avrdu: writing flash (2622 bytes):
Writing | #################################################################### | 100% 0.43s

(continues on next page)
Contents

• pio remote test
  – Usage
  – Description
  – Options
  – Examples

Usage

pio remote test [OPTIONS]

# run tests on specified PlatformIO Remote Agent
pio remote --agent NAME test [OPTIONS]

Description

Run remotely tests from PlatformIO based project. More details about PlatformIO Unit Testing.

This command allows you to apply the tests for the environments specified in "platformio.ini" (Project Configuration File).

Options

-e, --environment
Process specified environments. More details `pio run --environment`

--ignore

Ignore tests where the name matches specified patterns. More than one pattern is allowed. If you need to ignore some tests for the specific environment, please take a look at `test_ignore` option from “platformio.ini” (Project Configuration File).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches everything</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
<tr>
<td>[seq]</td>
<td>matches any character in seq</td>
</tr>
<tr>
<td>![seq]</td>
<td>matches any character not in seq</td>
</tr>
</tbody>
</table>

For example, `pio remote test --ignore "mytest*" -i "test[13]"

--upload-port

A port that is intended for firmware uploading. To list available ports please use `pio device list` command.

If upload port is not specified, PlatformIO will try to detect it automatically.

--test-port

A Serial/UART port that PlatformIO uses as communication interface between PlatformIO Unit Test Engine and target device. To list available ports please use `pio device list` command.

If test port is not specified, PlatformIO will try to detect it automatically.

-d, --project-dir

Specify the path to project directory. By default, `--project-dir` is equal to current working directory (CWD).

-r, --force-remote

By default, `Remote Development` processes project on a host machine and deploy final testing firmware (program) to remote device (embedded board).

If you need to process project on remote machine, please use `pio remote test --force-remote` option. In this case, `Remote Development` will automatically synchronize your project with remote machine, install required toolchains, frameworks, SDKs, etc., and process project.

--without-building

Skip building stage.

--without-uploading

Skip uploading stage

-v, --verbose

Shows detailed information when processing environments.

This option can also be set globally using `force_verbose` setting or by environment variable `PLATFORMIO_SETTING_FORCE_VERBOSE`.

Examples

For the examples please follow to `Unit Testing` page.
pio remote update

Usage

```bash
pio remote update [OPTIONS]
```

# start update process on the specified agents/machines
```bash
pio remote --agent NAME update [OPTIONS]
```

Description

Check or update installed Development Platforms and global Libraries on the remote machine.

Options

- `-c`, `--only-check`
  DEPRECATED. Please use `--dry-run` instead.

- `--dry-run`
  Do not update, only check for the new versions

Examples

```bash
> pio remote update
Platform Manager
=============
Platform timsp430
--------
Updating timsp430 @ 0.0.0: [Up-to-date]
Updating toolchain-timsp430 @ 1.40603.0: [Up-to-date]
Updating framework-energiams430 @ 1.17.0: [Up-to-date]
Updating framework-arduinomsp430 @ 1.10601.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Platform freescalekinetis
--------
```
Updating freescalekinetis @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform ststm32
--------
Updating ststm32 @ 0.0.0: [Up-to-date]
Updating framework-libopencm3 @ 1.1.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-stlink @ 1.10200.0: [Up-to-date]
Updating framework-spl @ 1.10201.0: [Up-to-date]
Updating framework-cmsis @ 1.40300.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform lattice_ice40
--------
Updating lattice_ice40 @ 0.0.0: [Up-to-date]
Updating toolchain-icestorm @ 1.7.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform atmelavr
--------
Updating atmelavr @ 0.0.0: [Up-to-date]
Updating framework-arduinoavr @ 1.10608.1: [Up-to-date]
Updating tool-avrdude @ 1.60001.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform espressif8266
--------
Updating espressif8266 @ 0.0.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-xtensena @ 1.40802.0: [Up-to-date]
Updating tool-esptool @ 1.409.0: [Up-to-date]
Updating tool-mksiffies @ 1.102.0: [Up-to-date]
Updating framework-arduinotespressif8266 @ 1.20300.0: [Up-to-date]
Updating sdk-esp8266 @ 1.10502.0: [Up-to-date]

Platform linux_x86_64
--------
Updating linux_x86_64 @ 0.0.0: [Up-to-date]
Updating toolchain-gcclinux64 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform windows_x86
--------
Updating windows_x86 @ 0.0.0: [Up-to-date]
Updating toolchain-gccmingw32 @ 1.40800.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform teensy
--------
Updating teensy @ 0.0.0: [Up-to-date]
Updating framework-arduinoteensy @ 1.128.0: [Up-to-date]
Updating tool-arduineteesy @ 1.1.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]

Platform nordicnrf51
--------
Updating nordicnrf51 @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating framework-arduinonordicnrf51 @ 1.20302.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform titiva
--------
Updating titiva @ 0.0.0: [Up-to-date]
Updating framework-libopencm3 @ 1.1.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating framework-energiativa @ 1.17.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform atmelsam
--------
Updating atmelsam @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-openocd @ 1.900.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating tool-avrdude @ 1.60001.1: [Up-to-date]
Updating tool-bossac @ 1.10601.0: [Up-to-date]

Platform siliconlabsefm32
--------
Updating siliconlabsefm32 @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform microchippic32
--------
Updating microchippic32 @ 0.0.0: [Up-to-date]
Updating framework-arduinomicrochippic32 @ 1.10201.0: [Up-to-date]
Updating toolchain-microchippic32 @ 1.40803.0: [Up-to-date]
Updating tool-pic32prog @ 1.200200.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform linux_i686
--------
Updating linux_i686 @ 0.0.0: [Up-to-date]
Updating toolchain-gccclinux32 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform intel.arc32
--------
Updating intel.arc32 @ 0.0.0: [Up-to-date]
Updating framework-arduinointel @ 1.10006.0: [Up-to-date]
Updating tool-arduino101load @ 1.124.0: [Up-to-date]
Updating toolchain-intelarc32 @ 1.40805.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform nxplpc
--------
Updating nxplpc @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform linux_arm
--------
Updating linux_arm @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmlinuxgernueabi @ 1.40802.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform native
--------
Updating native @ 0.0.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Library Manager
===============
Updating Adafruit-GFX @ 334e815bc1: [Up-to-date]
Updating Adafruit-ST7735 @ d53d4bf03a: [Up-to-date]
Updating Adafruit-DHT @ 09344416d2: [Up-to-date]
Updating Adafruit-Unified-Sensor @ f2af6f4e8c: [Up-to-date]
Updating ESP8266_SSD1306 @ 3.2.3: [Up-to-date]
Updating EngduinoMagnetometer @ 3.1.0: [Up-to-date]
Updating IRremote @ 2.2.1: [Up-to-date]
Updating Json @ 5.6.4: [Up-to-date]
Updating MODSERIAL @ d8422e6f3d: [Up-to-date]
git version 2.7.4 (Apple Git-66)
Already up-to-date.
Updating Servo @ 36b69a7ced07: [Checking]
Mercurial Distributed SCM (version 3.8.4)
(see https://mercurial-scm.org for more information)

Copyright (C) 2005-2016 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
pulling from https://developer.mbed.org/users/simon/code/Servo/
searching for changes
no changes found
Updating TextLCD @ 308d188a2d3a: [Checking]
Mercurial Distributed SCM (version 3.8.4)
(see https://mercurial-scm.org for more information)

Copyright (C) 2005-2016 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
pulling from https://developer.mbed.org/users/simon/code/TextLCD/
searching for changes
no changes found
pio run

Contents

• pio run
  – Usage
  – Description
  – Options
  – Examples

Usage

pio run [OPTIONS]

Description

Run project targets over environments declared in “platformio.ini” (Project Configuration File).

Options

-e, --environment
Process specified environments. Multiple environments are allowed.
You can also specify which environments should be processed by default using default_envs option from “platformio.ini” (Project Configuration File).

-t, --target
Process specified targets. Multiple targets are allowed.
You can configure default targets per project environment using targets option in “platformio.ini” (Project Configuration File).

--list-targets
New in version 5.0.
List available project targets. It’s also possible to list targets per project environment using pio run --environment option.
There are also built-in system targets:

  • Device
    – monitor automatically start pio device monitor after successful build operation. You can configure monitor using Monitor options.
  • System
    – envdump dump current build environment

--upload-port
Custom upload port of embedded board. To print all available ports use `pio device list` command.

If upload port is not specified, PlatformIO will try to detect it automatically.

`-d, --project-dir`
Specify the path to project directory. By default, `--project-dir` is equal to current working directory (CWD).

`-c, --project-conf`
Process project with a custom "platformio.ini" (Project Configuration File).

`-j, --jobs`
Control a number of parallel build jobs. Default is a number of CPUs in a system.

`-s, --silent`
Suppress progress reporting

`-v, --verbose`
Shows detailed information when processing environments.

This option can also be set globally using `force_verbose` setting or by environment variable `PLATFORMIO_SETTING_FORCE_VERBOSE`.

`--disable-auto-clean`
Disable auto-clean of `build_dir` when "platformio.ini" (Project Configuration File) or `src_dir` (project structure) have been modified.

Examples

1. Process Wiring Blink Example

```shell
> pio run

```

(continues on next page)
Program: 1034 bytes (3.2% Full) (.text + .data + .bootloader)

Data: 9 bytes (0.4% Full) (.data + .bss + .noinit)

========================================================================== [SUCCESS] Took 2.47 seconds ===========================================================================

--------------------------------------------------------------------------------------
Verbose mode can be enabled via `-v, --verbose` option
Collected 34 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/nodemcu/src/main.o
Archiving .pio/build/nodemcu/libFrameworkArduinoVariant.a
Indexing .pio/build/nodemcu/libFrameworkArduinoVariant.a
Compiling .pio/build/nodemcu/FrameworkArduino/FS.o
Compiling .pio/build/nodemcu/FrameworkArduino/HardwareSerial.o
... Archiving .pio/build/nodemcu/libFrameworkArduino.a
Indexing .pio/build/nodemcu/libFrameworkArduino.a
Linking .pio/build/nodemcu/firmware.elf
Calculating size .pio/build/nodemcu/firmware.elf
text data bss dec hex filename
221240 888 29400 251528 3d688 .pio/build/nodemcu/firmware.elf
Building .pio/build/nodemcu/firmware.bin
========================================================================== [SUCCESS] Took 6.43 seconds ===========================================================================

--------------------------------------------------------------------------------------
Verbose mode can be enabled via `-v, --verbose` option
Collected 96 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/teensy31/src/main.o
Compiling .pio/build/teensy31/FrameworkArduino/AudioStream.o
Compiling .pio/build/teensy31/FrameworkArduino/DMAChannel.o
... Compiling .pio/build/teensy31/FrameworkArduino/yield.o
Archiving .pio/build/teensy31/libFrameworkArduino.a
Indexing .pio/build/teensy31/libFrameworkArduino.a
Linking .pio/build/teensy31/firmware.elf
Calculating size .pio/build/teensy31/firmware.elf
text data bss dec hex filename
11288 168 2288 13744 35b0 .pio/build/teensy31/firmware.elf
Building .pio/build/teensy31/firmware.hex
========================================================================== [SUCCESS] Took 5.36 seconds ===========================================================================

Verbose mode can be enabled via `-v, --verbose` option
Collected 29 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/lpmsp430g2553/src/main.o
Compiling .pio/build/lpmsp430g2553/FrameworkAnergia/HardwareSerial.o
Compiling .pio/build/lpmsp430g2553/FrameworkAnergia/IPAddress.o
... Compiling .pio/build/lpmsp430g2553/FrameworkAnergia/wiring_digital.o
Compiling .pio/build/lpmsp430g2553/FrameworkAnergia/wiring_pulse.o
Compiling .pio/build/lpmsp430g2553/FrameworkAnergia/wiring_shift.o
Archiving .pio/build/lpmsp430g2553/libFrameworkAnergia.a
Indexing .pio/build/lpmsp430g2553/libFrameworkAnergia.a
Linking .pio/build/lpmsp430g2553/firmware.elf
Calculating size .pio/build/lpmsp430g2553/firmware.elf
text data bss dec hex filename
820 0 20 840 348 .pio/build/lpmsp430g2553/firmware.elf
Building .pio/build/lpmsp430g2553/firmware.hex

============== [SUCCESS] Took 2.34 seconds ==============

2. Process specific environment

> pio run -e nodemcu -e teensy31


Verbose mode can be enabled via `-v, --verbose` option
Collected 34 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/nodemcu/src/main.o
Archiving .pio/build/nodemcu/libFrameworkArduinoVariant.a
Indexing .pio/build/nodemcu/libFrameworkArduinoVariant.a
Compiling .pio/build/nodemcu/FrameworkArduino/FS.o
Compiling .pio/build/nodemcu/FrameworkArduino/HardwareSerial.o
... Archiving .pio/build/nodemcu/libFrameworkArduino.a
Indexing .pio/build/nodemcu/libFrameworkArduino.a
Linking .pio/build/nodemcu/firmware.elf
calculating size .pio/build/nodemcu/firmware.elf
text data bss dec hex filename
221240 888 29400 251528 3d688 .pio/build/nodemcu/firmware.elf
Building .pio/build/nodemcu/firmware.hex

============== [SUCCESS] Took 6.43 seconds ==============


Verbose mode can be enabled via `-v, --verbose` option
Collected 96 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/teensy31/src/main.o
Compiling .pio/build/teensy31/FrameworkArduino/AudioStream.o
Compiling .pio/build/teensy31/FrameworkArduino/DMAChannel.o
... Compiling .pio/build/teensy31/FrameworkArduino/yield.o
Archiving .pio/build/teensy31/libFrameworkArduino.a
Indexing .pio/build/teensy31/libFrameworkArduino.a
Linking .pio/build/teensy31/firmware.elf
Calculating size .pio/build/teensy31/firmware.elf
text data bss dec hex filename
11288 168 2288 13744 35b0 .pio/build/teensy31/firmware.elf
Building .pio/build/teensy31/firmware.hex

3. Process specific target (clean project)

> pio run -t clean


Removed .pio/build/uno/firmware.elf
Removed .pio/build/uno/firmware.hex
Removed .pio/build/uno/libFrameworkArduino.a
Removed .pio/build/uno/libFrameworkArduinoVariant.a
Removed .pio/build/uno/FrameworkArduino/_wiring_pulse.o
Removed .pio/build/uno/FrameworkArduino/abi.o
Removed .pio/build/uno/FrameworkArduino/CDC.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial10.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial11.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial12.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial13.o
Removed .pio/build/uno/FrameworkArduino/hooks.o
Removed .pio/build/uno/FrameworkArduino/IPAddress.o
Removed .pio/build/uno/FrameworkArduino/main.o
Removed .pio/build/uno/FrameworkArduino/new.o
Removed .pio/build/uno/FrameworkArduino/PluggableUSB.o
Removed .pio/build/uno/FrameworkArduino/Print.o
Removed .pio/build/uno/FrameworkArduino/Stream.o
Removed .pio/build/uno/FrameworkArduino/Tone.o
Removed .pio/build/uno/FrameworkArduino/USBCore.o
Removed .pio/build/uno/FrameworkArduino/WinInterrupts.o
Removed .pio/build/uno/FrameworkArduino/wiring.o
Removed .pio/build/uno/FrameworkArduino/wiring_analog.o
Removed .pio/build/uno/FrameworkArduino/wiring_analog.o
Removed .pio/build/uno/FrameworkArduino/wiring_analog.o
Removed .pio/build/uno/FrameworkArduino/wiring_analog.o
Removed .pio/build/uno/FrameworkArduino/wiring_analog.o
Removed .pio/build/uno/FrameworkArduino/wiring_shift.o
Removed .pio/build/uno/FrameworkArduino/WMath.o
Removed .pio/build/uno/FrameworkArduino/WSString.o
Removed .pio/build/uno/src/main.o
Done cleaning

removed .pio/build/nodemcu/firmware.bin
removed .pio/build/nodemcu/firmware.elf
removed .pio/build/nodemcu/libFrameworkArduino.a
removed .pio/build/nodemcu/libFrameworkArduinoVariant.a
...removed .pio/build/nodemcu/FrameworkArduino/spiffs/spiffs_nucleus.o
removed .pio/build/nodemcu/FrameworkArduino/umm_malloc/umm_malloc.o
removed .pio/build/nodemcu/src/main.o
done cleaning
====================== [SUCCESS] Took 0.50 seconds ======================

removed .pio/build/teensy31/firmware.elf
removed .pio/build/teensy31/firmware.hex
removed .pio/build/teensy31/libFrameworkArduino.a
removed .pio/build/teensy31/FrameworkArduino/analog.o
removed .pio/build/teensy31/FrameworkArduino/Atomic.o
...removed .pio/build/teensy31/FrameworkArduino/WString.o
removed .pio/build/teensy31/FrameworkArduino/yield.o
removed .pio/build/teensy31/src/main.o
done cleaning
====================== [SUCCESS] Took 0.50 seconds ======================

removed .pio/build/lpmsp430g2553/firmware.elf
removed .pio/build/lpmsp430g2553/firmware.hex
removed .pio/build/lpmsp430g2553/libFrameworkAnergia.a
removed .pio/build/lpmsp430g2553/FrameworkAnergia/atof.o
...removed .pio/build/lpmsp430g2553/FrameworkAnergia/avr/dtostrf.o
removed .pio/build/lpmsp430g2553/src/main.o
done cleaning
====================== [SUCCESS] Took 0.49 seconds ======================

4. Mix environments and targets

> pio run -e uno -t upload


verbose mode can be enabled via `--verbose` option
collected 36 compatible libraries
looking for dependencies...
project does not have dependencies
compiling .pio/build/uno/src/main.o
archiving .pio/build/uno/libFrameworkArduinoVariant.a
Indexing .pio/build/uno/libFrameworkArduinoVariant.a
Compiling .pio/build/uno/FrameworkArduino/CDC.o
...  
Compiling .pio/build/uno/FrameworkArduino/wiring_shift.o
Archiving .pio/build/uno/libFrameworkArduino.a
Indexing .pio/build/uno/libFrameworkArduino.a
Linking .pio/build/uno/firmware.elf
Checking program size .pio/build/uno/firmware.elf
text  data  bss  dec  hex filename
   1034    0    9    1043    413  .pio/build/uno/firmware.elf
Building .pio/build/uno/firmware.hex
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA141
Uploading .pio/build/uno/firmware.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e950f
avrdude: reading input file ".pio/build/uno/firmware.hex"
avrdude: writing flash (1034 bytes):

Writing | ################################################## | 100% 0.18s

avrdude: 1034 bytes of flash written
avrdude: verifying flash memory against .pio/build/uno/firmware.hex:
avrdude: load data flash data from input file .pio/build/uno/firmware.hex:
avrdude: input file .pio/build/uno/firmware.hex contains 1034 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.15s

avrdude: verifying ...
avrdude: 1034 bytes of flash verified

avrdude: safemode: Fuses OK (H:00, E:00, L:00)
avrdude done. Thank you.

================================= [SUCCESS] Took 4.14 seconds =================================

pio settings

Manage PlatformIO settings

Contents

- pio settings
  - pio settings get
    * Usage
    * Description
**Settings**

- `auto_update_libraries`
- `auto_update_platforms`
- `check_libraries_interval`
- `check_platformio_interval`
- `check_platforms_interval`
- `enable_cache`
- `enable_telemetry`
- `force.verbose`
- `projects_dir`

**Examples**

- `pio settings set`
- `pio settings reset`

**Usage**

```bash
pio settings get [NAME]
```

**Description**

**Note:**

- The *Yes* value is equal to: *True*, *Y*, *1* and is not case sensitive.
- You can override these settings using `Environment variables`.

Get/List existing settings

**Settings**
**auto_update_libraries**

*Default* No

*Values* Yes/No

Automatically update libraries.

**auto_update_platforms**

*Default* No

*Values* Yes/No

Automatically update platforms.

**check_libraries_interval**

*Default* 7

*Values* Days (Number)

Check for the library updates interval.

**check_platformio_interval**

*Default* 3

*Values* Days (Number)

Check for the new PlatformIO interval.

**check_platforms_interval**

*Default* 7

*Values* Days (Number)

Check for the platform updates interval.

**enable_cache**

*Default* Yes

*Values* Yes/No

Enable caching for API requests and Library Manager.

**enable_telemetry**

*Default* Yes

*Values* Yes/No
Share minimal diagnostics and usage information to help us make PlatformIO better.
The source code of telemetry service is open source. You can make sure that we DO NOT SHARE PRIVATE information or source code of your project. All information shares ANONYMously.

Which data do we collect and why?

- **A version of Python Interpreter.** PlatformIO Core (CLI) is written in Python language, including development Development Platforms. We need to know which Python version produces such type of exceptions (see below), which is more popular, which version we should drop and focus on a new one

- **PlatformIO Core (CLI) errors/exceptions.** We report automatically fatal exceptions raised by PlatformIO Core source code but NOT by your project

- **The name of the used platform, board, framework.** We collect this type of information to have a clear picture which software products are the most widely used by our Community and for the which we should provide frequent updates and add new features (for example, “atmelavr”, “arduino”, “uno”, etc.)

- **The name of CLI command.** It helps us to improve our CLI. For example, “run”, “lib list”)

- The name of **Cloud & Desktop IDE.** This is very important information for us. We create native extensions based on the popularity of IDEs (for example, VSCode, CLion)

Thanks a lot that you keep this setting enabled!

---

### force_verbose

**Default** No  
**Values** Yes/No

Force verbose output when processing environments. This setting overrides

- `pio run --verbose`
- `pio ci --verbose`
- `pio test --verbose`

---

### projects_dir

**Default** ~/Documents/PlatformIO/Projects  
**Values** Path to folder

Default location for PlatformIO projects (PlatformIO Home)

---

### Examples

1. List all settings and theirs current values

```
> pio settings get
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value [Default]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_update_libraries</td>
<td>No</td>
<td>Automatically update libraries (Yes/No)</td>
</tr>
</tbody>
</table>

(continues on next page)
### 2. Show specified setting

```bash
> pio settings get auto_update_platforms
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value [Default]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_update_platforms</td>
<td>Yes</td>
<td>Automatically update platforms (Yes/No)</td>
</tr>
</tbody>
</table>

### pio settings set

**Usage**

```
pio settings set NAME VALUE
```

**Description**

Set new value for the setting

**Examples**

Change to check for the new PlatformIO each day

```bash
> pio settings set check_platformio_interval 1
```

The new value for the setting has been set!

<table>
<thead>
<tr>
<th>Name</th>
<th>Value [Default]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>check_platformio_interval</td>
<td>1</td>
<td>Check for the new PlatformIO each day</td>
</tr>
</tbody>
</table>
pio settings reset

Usage

```plaintext
pio settings reset
```

Description

Reset settings to default

Examples

```bash
> pio settings reset
The settings have been reset!
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_update_libraries</td>
<td>No</td>
<td></td>
<td>Automatically update libraries (Yes/No)</td>
</tr>
<tr>
<td>auto_update_platforms</td>
<td>No</td>
<td></td>
<td>Automatically update platforms (Yes/No)</td>
</tr>
<tr>
<td>check_libraries_interval</td>
<td>7</td>
<td></td>
<td>Check for the library updates (interval in days)</td>
</tr>
<tr>
<td>check_platformio_interval</td>
<td>3</td>
<td></td>
<td>Check for the new PlatformIO (interval in days)</td>
</tr>
<tr>
<td>check_platforms_interval</td>
<td>7</td>
<td></td>
<td>Check for the platform updates (interval in days)</td>
</tr>
<tr>
<td>enable_cache</td>
<td>Yes</td>
<td></td>
<td>Enable caching for API requests and Library Manager</td>
</tr>
<tr>
<td>strict_ssl</td>
<td>No</td>
<td></td>
<td>Enable SSL for PlatformIO Services</td>
</tr>
<tr>
<td>enable_telemetry</td>
<td>Yes</td>
<td></td>
<td>Telemetry service? Yes/No</td>
</tr>
<tr>
<td>force_verbose</td>
<td>No</td>
<td></td>
<td>Force verbose output when processing environments</td>
</tr>
<tr>
<td>projects_dir</td>
<td>~/Documents/PlatformIO/Projects</td>
<td>Default location for PlatformIO projects (PlatformIO Home)</td>
<td></td>
</tr>
</tbody>
</table>

pio system

Miscellaneous system commands.

To print all available commands and options use:

```plaintext
pio system --help
pio system COMMAND --help
```

PlatformIO Shell Completion

Shell completion support for
To print all available commands and options use:

```bash
pio system completion --help
pio system completion COMMAND --help
```

### `pio system completion install`

#### Usage

```bash
pio system completion install [OPTIONS]
```

#### Description

Install shell completion files or code.

#### Options

`--shell`

The shell type, default is `auto` and will be detected from a current shell session.

Supported shells are:

- fish
- zsh
- bash
- powershell

`--path`

Custom installation path of the code to be evaluated by the shell. The standard installation path is used by default.
Examples

> pio system completion install

PlatformIO CLI completion has been installed for fish shell to ~/.config/fish/→completions/pio.fish
Please restart a current shell session

pio system completion uninstall

Contents

- pio system completion uninstall
  - Usage
  - Description
    * Options
  - Examples

Usage

pio system completion uninstall [OPTIONS]

Description

Uninstall shell completion files or code.

Options

--shell
The shell type, default is auto and will be detected from a current shell session.
Supported shells are:
- fish
- zsh
- bash
- powershell

--path
Custom installation path of the code to be evaluated by the shell. The standard installation path is used by default.
Examples

```bash
> pio system completion uninstall
PlatformIO CLI completion has been uninstalled for fish shell from ~/.config/fish/completions/pio.fish
Please restart a current shell session.
```

pio system info

New in version 5.0.

## Contents

- pio system info
  - Usage
  - Description
  - Options
  - Examples

## Usage

`pio system info`

## Description

Display PlatformIO system-wide information

## Options

**--json-output**

Return the output in JSON format

## Examples

```bash
> pio system info

PlatformIO Core: 5.0.0
Python: 3.8.5-final.0
System Type: darwin_x86_64
Platform: macOS-10.15.6
File System Encoding: utf-8
Locale Encoding: UTF-8
```

(continues on next page)
pio system prune

New in version 5.0.

Contents

- pio system prune
  - Usage
  - Description
  - Options
  - Examples

Usage

```shell
pio system prune
```

Description

Remove unused data:

- cached API requests
- cached package downloads
- temporary data.

Options

`--force, -f`

Do not prompt for confirmation.

Examples

```shell
> pio system prune
WARNING! This will remove:
- cached API requests
```
cached package downloads
- temporary data
Do you want to continue? [y/N]: y
Total reclaimed space: 36.48KB

pio team

New in version 5.0.

Manage organization teams and team memberships.

Management of organizations and owners is done with the `pio org` command.

To print all available commands and options use:

```
pio org --help
pio org COMMAND --help
```

pio team add

New in version 5.0.

Contents

- pio team add
  - Usage
  - Description
  - Examples
  - See Also

Usage

```
pio team add ORGNAME:TEAMNAME USERNAME
```

Description

Add a new member to a team.

If you need to create a new team, please use `pio team create` command.

Examples

Add Bob to the “developers” team of “platformio” organization

```
> pio team add platformio:developers bob
The new member "bob" has been successfully added to the "developers" team.
```
See Also

- pio team create
- pio team list
- pio team remove

pio team create

New in version 5.0.

Usage

`pio team create [OPTIONS] ORGNAME:TEAMNAME`

Description

Create a new team.

Options

--description

A team description.

Examples

```
> pio team create platformio:dev --description "Developers team"
The team "dev" has been successfully created.
```

See Also

- pio team list
- pio team remove
pio team destroy

New in version 5.0.

Contents

- pio team destroy
  - Usage
  - Description
  - See Also

Usage

pio team destroy ORGNAME:TEAMNAME

Description

Destroy a team.

If you need to remove a user from a team, please use pio team remove command.

See Also

- pio team create
- pio team list
- pio team update

pio team list

New in version 5.0.

Contents

- pio team list
  - Usage
  - Description
    - Options
  - Examples
Usage

```plaintext
pio team list [OPTIONS] [ORGNAME]
```

Description

List teams and their members.

Options

```
--json-output
```

Return the output in JSON format.

Examples

```
> pio team list

platformio:dev
------------------------
Description: Developers team
Members: alice, bob
```

### pio team remove

New in version 5.0.

Usage

```plaintext
pio team remove ORGNAME:TEAMNAME USERNAME
```

Description

Remove a member from a team.

If you need to destroy an existing team, please use `pio team destroy` command.
Examples

Remove Bob from the “dev” team of “platformio” organization:

```
> pio team remove platformio:dev bob
The "bob" member has been successfully removed from the "dev" team.
```

See Also

- `pio team add`
- `pio team list`

pio team update

New in version 5.0.

## Contents

- `pio team update`
  - Usage
  - Description
    - Options
  - Examples
  - See Also

## Usage

```
pio team update [OPTIONS] ORGNAME:TEAMNAME
```

## Description

Rename a team or update the existing details.

## Options

- **--name**
  
  A new team name. Team name must only contain alphanumeric characters, single hyphens, underscores, spaces. It can not begin or end with a hyphen or a underscore and must not be longer than 20 characters.

- **--description**
  
  A team description.
Examples

```shell
> pio team update platformio:dev --description "Developers team"
The team "dev" has been successfully updated.
```

See Also

- `pio team list`
- `pio team remove`

### pio test

Helper command for local *Unit Testing*.

#### Contents

- `pio test`
  - Usage
  - Description
  - Options
  - Examples

#### Usage

```
pio test [OPTIONS]
```

#### Description

Run locally tests from PlatformIO based project. More details about PlatformIO *Unit Testing*.

This command allows you to apply the tests for the environments specified in “platformio.ini” (*Project Configuration File)*.

#### Options

- `-e, --environment`
  
  Process specified environments. More details `pio run --environment`

- `-f, --filter`
  
  Process only the tests where the name matches specified patterns. More than one pattern is allowed. If you need to filter some tests for a specific environment, please take a look at `test_filter` option from “platformio.ini” (*Project Configuration File)*.
Pattern | Meaning
---|---
* | matches everything
? | matches any single character
[seq] | matches any character in seq
[!seq] | matches any character not in seq

For example, `pio test --filter "mytest*" -i "test[13]"

--ignore

Ignore tests where the name matches specified patterns. More than one pattern is allowed. If you need to ignore some tests for a specific environment, please take a look at `test_ignore` option from "platformio.ini" (Project Configuration File).

For example, `pio test --ignore "mytest*" -i "test[13]"

--upload-port

A port that is intended for firmware uploading. To list available ports please use `pio device list` command. If upload port is not specified, PlatformIO will try to detect it automatically.

--test-port

A Serial/UART port that PlatformIO uses as communication interface between PlatformIO Unit Test Engine and target device. To list available ports please use `pio device list` command. If test port is not specified, PlatformIO will try to detect it automatically.

-d, --project-dir

Specify the path to project directory. By default, `--project-dir` is equal to current working directory (CWD).

-c, --project-conf

Process project with a custom "platformio.ini" (Project Configuration File).

--without-building

Skip building stage.

--without-uploading

Skip uploading stage.

--without-testing

Skip testing stage.

--no-reset

Disable software reset via Serial.DTR/RST before test running. In this case, need to press "reset" button manually after firmware uploading.
Warning: If board does not support software reset via `Serial.DTR/RTS` you should add >2 seconds delay before `UNITY_BEGIN()`. We need that time to establish a `Serial` communication between host machine and target device. See Unit Testing.

`--monitor-rts`
Set initial `RTS` line state for Serial Monitor (0 or 1), default 1. We use it to gather test results via Serial connection.

`--monitor-dtr`
Set initial `DTR` line state for Serial Monitor (0 or 1), default 1. We use it to gather test results via Serial connection.

`-v, --verbose`
Shows detailed information when processing environments.
This option can also be set globally using `force_verbose` setting or by environment variable `PLATFORMIO_SETTING_FORCE_VERBOSE`.

Examples
For the examples please follow to Unit Testing page.

pio update

Contents

- pio update
  - Usage
  - Description
  - Options
  - Examples

Usage

```bash
pio update [OPTIONS]
```

Description

Check or update installed PlatformIO Core packages, Development Platforms and global Libraries. This command is combination of 2 sub-commands:

- `pio platform update`
- `pio lib update`

1.3. PlatformIO Core (CLI)
Options

--core-packages
Update only the core packages

-c, --only-check
DEPRECATED. Please use --dry-run instead.

--dry-run
Do not update, only check for the new versions

Examples

```bash
> pio update

Platform Manager
===============
Platform timsp430
--------
Updating timsp430 @ 0.0.0: [Up-to-date]
Updating toolchain-timsp430 @ 1.40603.0: [Up-to-date]
Updating framework-energiamp430 @ 1.17.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform freescalekinetis
--------
Updating freescalekinetis @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform ststm32
--------
Updating ststm32 @ 0.0.0: [Up-to-date]
Updating framework-libopencm3 @ 1.1.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-stlink @ 1.10200.0: [Up-to-date]
Updating framework-spl @ 1.10201.0: [Up-to-date]
Updating framework-cmsis @ 1.40300.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform lattice_ice40
--------
Updating lattice_ice40 @ 0.0.0: [Up-to-date]
Updating toolchain-icestorm @ 1.7.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform atmelavr
--------
Updating atmelavr @ 0.0.0: [Up-to-date]
Updating framework-arduinoavr @ 1.10608.1: [Up-to-date]
Updating tool-avrdude @ 1.60001.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
```

(continues on next page)
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform espressif8266
--------
Updating espressif8266 @ 0.0.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-xtensa @ 1.40802.0: [Up-to-date]
Updating tool-esptool @ 1.409.0: [Up-to-date]
Updating tool-mkspiffs @ 1.102.0: [Up-to-date]
Updating framework-arduinoexpressif8266 @ 1.20300.0: [Up-to-date]
Updating sdk-esp8266 @ 1.10502.0: [Up-to-date]

Platform linux_x86_64
--------
Updating linux_x86_64 @ 0.0.0: [Up-to-date]
Updating toolchain-gcclinux64 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform windows_x86
--------
Updating windows_x86 @ 0.0.0: [Up-to-date]
Updating toolchain-gccmingw32 @ 1.40800.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform teensy
--------
Updating teensy @ 0.0.0: [Up-to-date]
Updating framework-arduinoteensy @ 1.128.0: [Up-to-date]
Updating tool-teensy @ 1.1.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]

Platform nordicnrf51
--------
Updating nordicnrf51 @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating framework-arduinonordicnrf51 @ 1.20302.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform titiva
--------
Updating titiva @ 0.0.0: [Up-to-date]
Updating framework-libopencm3 @ 1.1.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating framework-energiativa @ 1.17.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform atmelsam
--------
Updating atmelsam @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-openocd @ 1.900.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Platform siliconlabsefm32
--------
Updating siliconlabsefm32 @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform microchipp32
--------
Updating microchipp32 @ 0.0.0: [Up-to-date]
Updating framework-arduinomicrochipp32 @ 1.10201.0: [Up-to-date]
Updating toolchain-microchipp32 @ 1.40803.0: [Up-to-date]
Updating tool-pic32prog @ 1.200200.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform linux_i686
--------
Updating linux_i686 @ 0.0.0: [Up-to-date]
Updating toolchain-gcclinux32 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform intel_arc32
--------
Updating intel_arc32 @ 0.0.0: [Up-to-date]
Updating framework-arduinointel @ 1.10006.0: [Up-to-date]
Updating toolchain-intelarc32 @ 1.40805.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform nxplpc
--------
Updating nxplpc @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform linux_arm
--------
Updating linux_arm @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmlinuxgnuarm @ 1.40802.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform native
--------
Updating native @ 0.0.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Library Manager
===============
Updating Adafruit-GFX @ 334e815b0c1: [Up-to-date]
Updating Adafruit-ST7735 @ d53d4bf03a: [Up-to-date]
Updating Adafruit-DHT @ 0934416d2: [Up-to-date]
Updating Adafruit-Unified-Sensor @ f2af6f4efc: [Up-to-date]
Updating ESP8266_SSD1306 @ 3.2.3: [Up-to-date]
Updating EngduinoMagnetometer @ 3.1.0: [Up-to-date]
Updating IRremote @ 2.2.1: [Up-to-date]
Updating Json @ 5.6.4: [Up-to-date]
Updating MODSERIAL @ d8422e4e47: [Up-to-date]
Updating PJON @ 1fb26fd: [Checking]
git version 2.7.4 (Apple Git-66)
Already up-to-date.
Updating Servo @ 36b69a7ced07: [Checking]
Mercurial Distributed SCM (version 3.8.4)
(see https://mercurial-scm.org for more information)

Copyright (C) 2005-2016 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
pulling from https://developer.mbed.org/users/simon/code/Servo/
searching for changes
no changes found
Updating TextLCD @ 308d188a2d3a: [Checking]
Mercurial Distributed SCM (version 3.8.4)
(see https://mercurial-scm.org for more information)

Copyright (C) 2005-2016 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
pulling from https://developer.mbed.org/users/simon/code/TextLCD/
searching for changes
no changes found

pio upgrade

Contents

• pio upgrade
  – Usage
  – Description
  – Options
  – Examples

Usage

pio upgrade

Description

Check or upgrade PlatformIO to the latest version
Options

`--dev`
Use development branch.

Examples

```bash
> pio upgrade
You are up-to-date!
PlatformIO x.x.x is currently the newest version available.

# If you have problem with permissions try:
> sudo pio upgrade
```

1.4 PlatformIO Home

PlatformIO Home allows you to interact with PlatformIO’s professional collaborative platform using modern and cross-platform GUI:

- Project Manager
- PlatformIO Account
- Library Management
- Development Platforms
- Library and development platform updates
- Frameworks
- Boards
- Device Manager: serial, logical, and multicast DNS services
- Static Code Analysis
- Firmware File Explorer
- Firmware Memory Inspection
- Firmware Sections & Symbols Viewer.

Contents

- Installation
- Quick Start
  - PlatformIO IDE
  - PlatformIO Core
- Demo
  - Welcome & Project Manager
1.4.1 Installation

You do not need to install PlatformIO Home separately, it’s already built-in in PlatformIO IDE and PlatformIO Core (CLI).

1.4.2 Quick Start

PlatformIO IDE

Please open PlatformIO Home using (HOME) button on PlatformIO Toolbar:

- **Atom**: PlatformIO Toolbar
- **VSCode**: PlatformIO Toolbar

PlatformIO Core

Please launch PlatformIO Home Web-server using `pio home` command and open in your browser http://127.0.0.1:8008.

You can change host and port. Please check `pio home` command for details.

1.4.3 Demo
Project Inspect
Statistics

Top 5 Files

156.0 KB  unknown
22.8 KB  ...sonoff/xdrv_04_light.ino
21.5 KB  ...Sonoff-Tasmota/sonoff/sonoff.ino
17.1 KB  ...sonoff/xdrv_01_webserver.ino
12.4 KB  ...Deserialization/JsonParserimpl.hpp

Top 5 Symbols

9.4 KB  MqttDataHandler(char*, unsigned char*, unsigned int)
6.8 KB  HueLights(String*)
5.6 KB  LightCommand()
3.9 KB  web_log
3.5 KB  Settings

Top Defects

<table>
<thead>
<tr>
<th>Level</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
<td>The scope of the variable 'result' can be reduced.</td>
</tr>
<tr>
<td>LOW</td>
<td>The scope of the variable 'button_present' can be reduced.</td>
</tr>
<tr>
<td>LOW</td>
<td>Variable 'button' is assigned a value that is never used.</td>
</tr>
<tr>
<td>LOW</td>
<td>Variable 'button_present' is assigned a value that is never used.</td>
</tr>
</tbody>
</table>
Only code analysis (Static Code Analysis)

![Static Code Analysis](image-url)

**Defects Summary**

<table>
<thead>
<tr>
<th>Component</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>include</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>include/external</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>src</td>
<td>10</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>src/comms</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>src/hw/spi</td>
<td>4</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>src/hw/uart</td>
<td>1</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>src/misra</td>
<td>26</td>
<td>191</td>
<td>58</td>
</tr>
<tr>
<td>src/samples</td>
<td>24</td>
<td>165</td>
<td>58</td>
</tr>
<tr>
<td>src/sensors</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

**Top Defects**

<table>
<thead>
<tr>
<th>Level</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH</td>
<td>Array 'arr[10]' accessed at index 10, which is out of bounds.</td>
</tr>
<tr>
<td>HIGH</td>
<td>Memory pointed to by 'pl' is freed twice.</td>
</tr>
<tr>
<td>HIGH</td>
<td>Mismatching allocation and deallocation: pl</td>
</tr>
<tr>
<td>HIGH</td>
<td>Memory is allocated but not initialized: pl</td>
</tr>
<tr>
<td>HIGH</td>
<td>Resource leak: pMemory</td>
</tr>
</tbody>
</table>

If you enjoy using PlatformIO, please star our projects on GitHub!

★ PlatformIO Core ★
Firmware File Explorer

File Symbols
### Test PlatformIO/Sonoff-Tasmota

#### Environment: Sonoff

**ESP8266 80MHz, 80 KB RAM, 1,000.0 KB Flash**

### Inspector View

**Search:** For ex. “init 0x80 bss"

**Platforms**:
- SetLedLink(unsigned char)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40201888
  - **Size**: 77 bytes
- GpioInit()
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x402177B4
  - **Size**: 1.5 KB
- MqttPublishTeleState()
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x402210E4
  - **Size**: 36 bytes
- GetOtaUrl(char*, unsigned int)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x402812F4
  - **Size**: 110 bytes
- GetFallbackTopic_P(char*, unsigned char, char const*)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40281664
  - **Size**: 27 bytes
- SendKey(unsigned char, unsigned char, unsigned char)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40220774
  - **Size**: 360 bytes
- GetPulseTimer(unsigned char)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40283698
  - **Size**: 80 bytes
- SetAllPower(unsigned char, int)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x4022038C
  - **Size**: 69 bytes
- SerialInput()
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40227884
  - **Size**: 624 bytes
- SetLedPower(unsigned char)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x402817D4
  - **Size**: 110 bytes
- setup
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x4021A228
  - **Size**: 1.1 KB
- PublishStatus(unsigned char)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x4022119C
  - **Size**: 2.0 KB
- MqttDataHandler(char*, unsigned char, char*, unsigned int)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40224A0C
  - **Size**: 9.4 KB
- GetTopic_P(char*, unsigned char, char*, char const*)
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40201388
  - **Size**: 684 bytes
- Every250mSeconds()
  - **Type**: STT_FUNC
  - **Bind**: STB_GLOBAL
  - **Address**: 0x40233988
  - **Size**: 1.3 KB

**Total Size on Page:** 17.5 KB
### Firmware Symbols

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Bind</th>
<th>Address</th>
<th>Section</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MqttDataHandler(char*, unsigned char*, unsigned int)</td>
<td>STT_FUNC</td>
<td>STB_GLOBAL</td>
<td>0x00224A0C</td>
<td>.text</td>
<td>9.4 KB</td>
</tr>
<tr>
<td>HueLights(String*)</td>
<td>STT_FUNC</td>
<td>STB_GLOBAL</td>
<td>0x002283E8</td>
<td>.text</td>
<td>6.8 KB</td>
</tr>
<tr>
<td>LightCommand()</td>
<td>STT_FUNC</td>
<td>STB_GLOBAL</td>
<td>0x00229354</td>
<td>.text</td>
<td>5.6 KB</td>
</tr>
<tr>
<td>web_log</td>
<td>STT_OBJECT</td>
<td>STB_GLOBAL</td>
<td>0x3FFF0208</td>
<td>.text</td>
<td>3.9 KB</td>
</tr>
<tr>
<td>Settings</td>
<td>STT_OBJECT</td>
<td>STB_GLOBAL</td>
<td>0x3FFF1C64</td>
<td>.text</td>
<td>3.5 KB</td>
</tr>
<tr>
<td>Arduino::Json::Internals::JsonParser&lt;Arduino::Json::Internals::CharPointerTraits&lt; char const&gt;*&gt;::Reader, Arduino::Json::Internals::StaticJsonBufferBase &amp;&gt;::parseObject()</td>
<td>STT_FUNC</td>
<td>STB_WEAK</td>
<td>0x00204888</td>
<td>.text</td>
<td>3.4 KB</td>
</tr>
<tr>
<td>Arduino::Json::Internals::JsonParser&lt; Arduino::Json::Internals::StdStringTraits&lt; String*&gt;&gt;::Reader, Arduino::Json::Internals::StaticJsonBufferBase &amp;&gt;::parseObject()</td>
<td>STT_FUNC</td>
<td>STB_WEAK</td>
<td>0x002129A8</td>
<td>.text</td>
<td>3.4 KB</td>
</tr>
</tbody>
</table>
### Firmware Sections

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Flags</th>
<th>Address</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.bss</td>
<td>SHT_NOBITS</td>
<td>WA</td>
<td>0x3FFE9938</td>
<td>39.8 KB</td>
</tr>
<tr>
<td>.comment</td>
<td>SHT_PROGBITS</td>
<td>WA</td>
<td>0x00000000</td>
<td>4.9 KB</td>
</tr>
<tr>
<td>.data</td>
<td>SHT_PROGBITS</td>
<td>WA</td>
<td>0x3FFE0000</td>
<td>1.7 KB</td>
</tr>
<tr>
<td>.from0.text</td>
<td>SHT_PROGBITS</td>
<td>AX</td>
<td>0x40201010</td>
<td>569.3 KB</td>
</tr>
<tr>
<td>.noinit</td>
<td>SHT_PROGBITS</td>
<td>WA</td>
<td>0x3FFE86D4</td>
<td>4 bytes</td>
</tr>
<tr>
<td>.rodata</td>
<td>SHT_PROGBITS</td>
<td>WA</td>
<td>0x3FFE86E8</td>
<td>4.8 KB</td>
</tr>
<tr>
<td>.shstrtab</td>
<td>SHT_STRTAB</td>
<td></td>
<td>0x00000000</td>
<td>16.7 KB</td>
</tr>
<tr>
<td>.strtab</td>
<td>SHT_STRTAB</td>
<td></td>
<td>0x00000000</td>
<td>144.2 KB</td>
</tr>
<tr>
<td>.symtab</td>
<td>SHT_SYMTAB</td>
<td></td>
<td>0x00000000</td>
<td>93.7 KB</td>
</tr>
<tr>
<td>.text</td>
<td>SHT_PROGBITS</td>
<td>AX</td>
<td>0x40300000</td>
<td>504 bytes</td>
</tr>
</tbody>
</table>
## Static Code Analysis

<table>
<thead>
<tr>
<th>Tool</th>
<th>Level</th>
<th>Category</th>
<th>Message</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>clangtidy</td>
<td>MEDIUM</td>
<td>WARNING</td>
<td>header is missing header guard</td>
<td>...include/external/ext.hpp:1:1</td>
</tr>
<tr>
<td>clangtidy</td>
<td>MEDIUM</td>
<td>WARNING</td>
<td>header is missing header guard</td>
<td>...include/main.hpp:1:1</td>
</tr>
<tr>
<td>clangtidy</td>
<td>MEDIUM</td>
<td>WARNING</td>
<td>both sides of operator are equivalent</td>
<td>...src/comms/protocol.cpp:8:20</td>
</tr>
<tr>
<td>cppcheck</td>
<td>MEDIUM</td>
<td>WARNING</td>
<td>CWE-398: Member variable 'mValue' is not initialized in the constructor.</td>
<td>...src/comms/protocol.cpp:16:5</td>
</tr>
<tr>
<td>clangtidy</td>
<td>MEDIUM</td>
<td>WARNING</td>
<td>constructor does not initialize these fields: mValue</td>
<td>...src/comms/protocol.cpp:16:5</td>
</tr>
<tr>
<td>clangtidy</td>
<td>MEDIUM</td>
<td>WARNING</td>
<td>use ‘= default’ to define a trivial default constructor</td>
<td>...src/comms/protocol.cpp:16:5</td>
</tr>
<tr>
<td>cppcheck</td>
<td>HIGH</td>
<td>ERROR</td>
<td>CWE-418: Memory pointed to by 'pl' is freed twice.</td>
<td>...src/hw/spi/spi.cpp:8:5</td>
</tr>
<tr>
<td>cppcheck</td>
<td>HIGH</td>
<td>ERROR</td>
<td>CWE-908: Memory is allocated but not initialized: pl</td>
<td>...src/hw/spi/spi.cpp:14:6</td>
</tr>
<tr>
<td>cppcheck</td>
<td>MEDIUM</td>
<td>WARNING</td>
<td>CWE-783: In expression like &quot;A++&quot;, the result of &quot;++&quot; is unused. Did you intend to write &quot;(A)++;&quot;?</td>
<td>...src/hw/spi/spi.cpp:14:8</td>
</tr>
</tbody>
</table>
Library Manager

Search libraries

Recently

Updated
- U8g2 3 hours ago
- esp8266ndn 3 hours ago
- Adafruit BME280 Library 12 hours ago
- IHCSoapClient 17 hours ago
- RapidJSON 20 hours ago

Added
- esp8266ndn 3 hours ago
- Cryptosuite 1 day ago
- CryptoC 1 day ago
- Crypto 1 day ago
- NTPtimeESP 1 day ago

Keywords
- sha1
- hmac
- upnp
- smartthings
- ssdp

Popular Tags

- display
- communication
- sensors
- control
- device
- lcd
- graphics
- tft
- oled
- displaycore
- glcd
- other
- input
- signal
- output
- data
- font
- sensor
- i2c
- spi
- timing
- esp8266
- processing
- storage
- serial
- wifi
- temperature
- http
- arduino
- iot
- rf
- led
- radio
- web
- i2cdevlib
- ethernet
- uncategorized
- time
- timer
- wireless
- mqtt
- mbed
- server
- protocol
- accelerometer
- wi-fi
- button
- rtc
- humidity
- rest

Trending

Today
- SerialESP8266wifi

Week
- PubSubClient

Month
- ArduinoJson
## Board Explorer

PlatformIO currently supports over 400 boards from leading manufacturers, and we are constantly adding new ones. You can be part of the process by letting us know what board you wish to see supported next, by submitting a feature request.

### Search board...

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Frameworks</th>
<th>MCU</th>
<th>FRQ</th>
<th>ROM</th>
<th>RAM</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>CMSIS, libOpenCM3, SPL, STM32Cube</td>
<td>STM32</td>
<td>168 Mhz</td>
<td>1 MB</td>
<td>128 KB</td>
<td><img src="https://example.com/icon.png" alt="Icon" /></td>
</tr>
<tr>
<td>4DSystems</td>
<td>MicrochipPIC32</td>
<td>Arduino</td>
<td>32MX7</td>
<td>508 KB</td>
<td>128 KB</td>
<td><img src="https://example.com/icon.png" alt="Icon" /></td>
<td></td>
</tr>
<tr>
<td>96Boards B</td>
<td>ST STM32</td>
<td>mbed, STM32Cube</td>
<td>STM32F446VE</td>
<td>168 Mhz</td>
<td>512 KB</td>
<td>128 KB</td>
<td><img src="https://example.com/icon.png" alt="Icon" /></td>
</tr>
<tr>
<td>Adafruit Blu</td>
<td>Atmel AVR</td>
<td>Arduino</td>
<td>ATMEGA32U4</td>
<td>8 Mhz</td>
<td>28 KB</td>
<td>2.5 KB</td>
<td><img src="https://example.com/icon.png" alt="Icon" /></td>
</tr>
<tr>
<td>Adafruit CircuitPlaygroundExpress</td>
<td>Atmel SAM</td>
<td>Arduino</td>
<td>SAMD21G18A</td>
<td>48 Mhz</td>
<td>256 KB</td>
<td>32 KB</td>
<td><img src="https://example.com/icon.png" alt="Icon" /></td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif32</td>
<td>Arduino, ESP-IDF</td>
<td>ESP32</td>
<td>240 Mhz</td>
<td>1 MB</td>
<td>288 KB</td>
<td><img src="https://example.com/icon.png" alt="Icon" /></td>
</tr>
</tbody>
</table>

## 1.5 Tutorials and Examples

### 1.5.1 Tutorials


Unit Testing of a “Blink” Project

The goal of this tutorial is to demonstrate how simple it is to use Unit Testing.

- **Level:** Beginner
- **Platforms:** Windows, macOS, Linux

## Contents

- Setting Up the Project
- Project structure
- Source files
- Test results

## Setting Up the Project

1. Please navigate to the Quick Start section and create the “Blink Project”.
2. Create a test directory in the project (on the same level as src) and place a test_main.cpp file in it (the source code is located below).
3. Run tests using the pio test command.

### Project structure

```
project_dir
  └── lib
      └── README
  └── platformio.ini
  └── src
      └── ...
  └── test
      └── test_main.cpp
```

### Source files

- **platformio.ini**

```ini
[env:uno]
```

(continues on next page)
platform = atmelavr
framework = arduino
board = uno

[env:teensy31]
platform = teensy
framework = arduino
board = teensy31

• test/test_main.cpp

#include <Arduino.h>
#include <unity.h>

// void setUp(void) {
//   // set stuff up here
// }

// void tearDown(void) {
//   // clean stuff up here
// }

void test_led_builtin_pin_number(void) {
    TEST_ASSERT_EQUAL(13, LED_BUILTIN);
}

void test_led_state_high(void) {
    digitalWrite(LED_BUILTIN, HIGH);
    TEST_ASSERT_EQUAL(HIGH, digitalRead(LED_BUILTIN));
}

void test_led_state_low(void) {
    digitalWrite(LED_BUILTIN, LOW);
    TEST_ASSERT_EQUAL(LOW, digitalRead(LED_BUILTIN));
}

void setup() {
    // NOTE!!! Wait for >2 secs
    // if board doesn't support software reset via Serial.DTR/RTS
    delay(2000);

    UNITY_BEGIN(); // IMPORTANT LINE!
    RUN_TEST(test_led_builtin_pin_number);
    pinMode(LED_BUILTIN, OUTPUT);
}

uint8_t i = 0;
uint8_t max_blinks = 5;

void loop() {
    if (i < max_blinks) {
        RUN_TEST(test_led_state_high);
        delay(500);
        RUN_TEST(test_led_state_low);
        delay(500);
    }
}
```c
i++;
}
else if (i == max_blinks) {
    UNITY_END(); // stop unit testing
}
```

Test results

```bash
> pio test -e uno --verbose

Verbose mode can be enabled via `--verbose` option
Collected 1 items

===================== [test/*] Building... (1/3) =======================
Processing uno (platform: atmelavr; board: uno; framework: arduino)

Collected 24 compatible libraries
Scanning dependencies...
No dependencies
Compiling .pio/build/uno/test/output_export.cpp.o
Compiling .pio/build/uno/test/test_main.cpp.o
Archiving .pio/build/uno/libFrameworkArduinoVariant.a
Compiling .pio/build/uno/FrameworkArduino/CDC.cpp.o
Indexing .pio/build/uno/libFrameworkArduinoVariant.a
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial1.cpp.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial10.cpp.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial11.cpp.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial12.cpp.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial13.cpp.o
Compiling .pio/build/uno/FrameworkArduino/IPAddress.cpp.o
Compiling .pio/build/uno/FrameworkArduino/PluggableUSB.cpp.o
Compiling .pio/build/uno/FrameworkArduino/Print.cpp.o
Compiling .pio/build/uno/FrameworkArduino/Stream.cpp.o
Compiling .pio/build/uno/FrameworkArduino/Tone.cpp.o
Compiling .pio/build/uno/FrameworkArduino/USBCore.cpp.o
Compiling .pio/build/uno/FrameworkArduino/WInterrupts.c.o
Compiling .pio/build/uno/FrameworkArduino/WMath.cpp.o
Compiling .pio/build/uno/FrameworkArduino/WString.cpp.o
Compiling .pio/build/uno/FrameworkArduino/abi.cpp.o
Compiling .pio/build/uno/FrameworkArduino/hooks.c.o
Compiling .pio/build/uno/FrameworkArduino/main.cpp.o
Compiling .pio/build/uno/FrameworkArduino/new.cpp.o
Compiling .pio/build/uno/FrameworkArduino/wiring.c.o
Compiling .pio/build/uno/FrameworkArduino/wiring_analog.c.o
Compiling .pio/build/uno/FrameworkArduino/wiring_digital.c.o
Compiling .pio/build/uno/FrameworkArduino/wiring_pulse.S.o
Compiling .pio/build/uno/FrameworkArduino/wiring_pulse.c.o
Compiling .pio/build/uno/FrameworkArduino/wiring_shift.c.o
```
Compiling .pio\build\uno\UnityTestLib\unity.o
Archiving .pio\build\uno\libFrameworkArduino.a
Indexing .pio\build\uno\libFrameworkArduino.a
Archiving .pio\build\uno\libUnityTestLib.a
Indexing .pio\build\uno\libUnityTestLib.a
Linking .pio\build\uno\firmware.elf
Checking size .pio\build\uno\firmware.elf
Building .pio\build\uno\firmware.hex
DATA: [== ] 20.0% (used 410 bytes from 2048 bytes)
PROGRAM: [= ] 12.6% (used 4060 bytes from 32256 bytes)

========================================== [SUMMARY]

Environment uno [SUCCESS]
Environment teensy31 [SKIP]
========================================== [SUCCESS] Took 2.54 seconds

================================= [test/*] Uploading... (2/3)

Processing uno (platform: atmelavr; board: uno; framework: arduino)

Verbose mode can be enabled via `--v, --verbose` option
PLATFORM: Atmel AVR > Arduino Uno
SYSTEM: ATMEGA328P 16MHz 2KB RAM (31.50KB Flash)
LDF MODES: FINDER(chain) COMPATIBILITY(soft)
Collected 24 compatible libraries
Scanning dependencies...
No dependencies
Checking size .pio\build\uno\firmware.elf
DATA: [== ] 20.0% (used 410 bytes from 2048 bytes)
PROGRAM: [= ] 12.6% (used 4060 bytes from 32256 bytes)
Configuring upload protocol...
AVAILABLE: arduino
CURRENT: upload_protocol = arduino
Looking for upload port...
Auto-detected: COM18
Uploading .pio\build\uno\firmware.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ************************************************************** | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file ".pio\build\uno\firmware.hex"
avrdude: writing flash (4060 bytes):

Writing | ************************************************************** | 100% 0.76s

avrdude: 4060 bytes of flash written
avrdude: verifying flash memory against .pio\build\uno\firmware.hex:
avrdude: load data flash data from input file .pio\build\uno\firmware.hex:
avrdude: input file .pio\build\uno\firmware.hex contains 4060 bytes
avrdude: reading on-chip flash data:

(continues on next page)
avrdude: verifying ...
avrdude: 4060 bytes of flash verified
avrdude: safemode: Fuses OK (E:00, H:00, L:00)
avrdude done. Thank you.

================================== [SUMMARY] ===============
Environment uno [SUCCESS]
Environment teensy31 [SKIP]
================================== [Success] Took 4.45 seconds ============

If you don’t see any output for the first 10 secs, please reset board (press reset button)

test\test_main.cpp:30:test_led_builtin_pin_number [PASSED]
test\test_main.cpp:41:test_led_state_high [PASSED]
test\test_main.cpp:43:test_led_state_low [PASSED]
test\test_main.cpp:41:test_led_state_high [PASSED]
test\test_main.cpp:43:test_led_state_low [PASSED]
test\test_main.cpp:41:test_led_state_high [PASSED]
test\test_main.cpp:43:test_led_state_low [PASSED]
test\test_main.cpp:41:test_led_state_high [PASSED]
test\test_main.cpp:43:test_led_state_low [PASSED]
test\test_main.cpp:41:test_led_state_high [PASSED]
test\test_main.cpp:43:test_led_state_low [PASSED]
test\test_main.cpp:41:test_led_state_high [PASSED]
test\test_main.cpp:43:test_led_state_low [PASSED]

11 Tests 0 Failures 0 Ignored

============================ [TEST SUMMARY] =========
test/*/env:uno [PASSED]
test/*/env:teensy31 [IGNORED]

Get started with Arduino and ESP32-DevKitC: debugging and unit testing

The goal of this tutorial is to demonstrate how simple it is to use VSCode to develop, run and debug a simple project with the Arduino framework for the ESP32-DevKitC board.

- **Level:** Beginner
- **Platforms:** Windows, Mac OS X, Linux

**Requirements:**

- Downloaded and installed VSCode
- Espressif ESP32 Dev Module
- Olimex ARM-USB-OCD or Olimex ARM-USB-TINY adapter for debugging
Setting Up the Project

First, we need to create a new project using the PlatformIO Home Page (to open this page, just press the Home icon on the toolbar):

Next, we need to select **Espressif ESP32 Dev Module** as a development board, **Arduino** as a framework and a path to the project location (or use the default one):
After that, we have a fully configured project that is ready for developing code with the Arduino framework.

Adding Code to the Generated Project

Let’s add some actual code to the project. Firstly, we open a default main file named `main.cpp` in the `src_dir` folder and replace its content with following:

```cpp
#include <Arduino.h>

void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.println("Hello world!");
    delay(1000);
}
```
We have now created a basic project ready for compiling and uploading.

**Compiling and Uploading the Firmware**

Now we can build the project. There are several ways to compile firmware:

- Build option in the **Project Tasks** menu,
- Build button in **PlatformIO Toolbar**,
- **Task Menu**: Tasks: Run Task... > PlatformIO: Build, or in the **PlatformIO Toolbar**,
- **Command Palette**: View: Command Palette > PlatformIO: Build, or
- via hotkeys cmd-alt-b / ctrl-alt-b

Marked in red:
If everything went well, we should see a Success message in the terminal window:
There are also several ways to upload the firmware to the board:

- Upload option in the Project Tasks menu,
- Upload button in PlatformIO Toolbar,
- Command Palette: View: Command Palette > PlatformIO: Upload,
- using the Task Menu: Tasks: Run Task... > PlatformIO: Upload, or
- via hotkeys: cmd-alt-u / ctrl-alt-u:
After uploading, we need to check if the firmware is uploaded correctly. To do this, open the serial monitor and check that the message from the board is received. To open the serial monitor, we can use the following options:

- Monitor option in the **Project Tasks** menu,
- Serial Monitor button in the **PlatformIO Toolbar**,
- **Command Palette**: View: Command Palette > PlatformIO: Monitor, or
- **Task Menu**: Tasks: Run Task... > PlatformIO: Monitor:
If the firmware works as expected, the message from the board can be observed in the terminal window:
Debugging the Firmware

Setting Up the Hardware

In order to use a JTAG probe with an ESP32, we need to connect the following pins:

<table>
<thead>
<tr>
<th>ESP32 pin</th>
<th>JTAG probe pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3V</td>
<td>Pin 1 (VTref)</td>
</tr>
<tr>
<td>GPIO 9 (EN)</td>
<td>Pin 3 (nTRST)</td>
</tr>
<tr>
<td>GND</td>
<td>Pin 4 (GND)</td>
</tr>
<tr>
<td>GPIO 12 (TDI)</td>
<td>Pin 5 (TDI)</td>
</tr>
<tr>
<td>GPIO 14 (TMS)</td>
<td>Pin 7 (TMS)</td>
</tr>
<tr>
<td>GPIO 13 (TCK)</td>
<td>Pin 9 (TCK)</td>
</tr>
<tr>
<td>GPIO 15 (TDO)</td>
<td>Pin 13 (TDO)</td>
</tr>
</tbody>
</table>

Debugging offers the easiest way to debug the board. Firstly, we need to specify debug_tool in “platformio.ini” (Project Configuration File). In this tutorial, an Olimex ARM-USB-OCD-H debug probe is used:

```
[env:esp32dev]
platform = espressif32
board = esp32dev
```

(continues on next page)
To start the debug session we can use the following methods:

- Debug: Start debugging in the top menu,
- Start Debugging option in the Quick Access menu, or
- hotkey button F5:

We need to wait some time while PlatformIO initializes the debug session, and are ready to debug when the first line after the main function is highlighted.

1. Please wait when debugging session is stopped at the first line of `app_main()` function
2. **WARNING!** Please set a breakpoint at `void loopTask(void *pvParameters)` (line 13 in the screenshot below - this line can change between releases)
3. Now, please press CONTINUE/RUN button on debugging toolbar (right arrow icon)
4. The debugging session should stop at the first line of the `void loopTask(void *pvParameters)` function
5. Now, navigate to your Arduino setup/loop code and do classic debugging.
We can walk through the code using control buttons, set breakpoints, and add variables to the Watch window:
Writing Unit Tests

Test cases can be added to a single file that may include multiple tests. First of all, in this file, we need to add four default functions: `setUp`, `tearDown`, `setup` and `loop`. Functions `setUp` and `tearDown` are used to initialize and finalize test conditions. Implementations of these functions are not required for running tests, but if you need to initialize some variables before you run a test, use the `setUp` function. Likewise, if you need to clean up variables, use `tearDown` function. In our example we will use these functions to respectively initialize and deinitialize LED states. The `setup` and `loop` functions act as a simple Arduino program where we describe our test plan.

Let’s create a `test` folder in the root of the project and add a new file, `test_main.cpp`, to this folder. Next, basic tests for `String` class will be implemented in this file:

- `test_string_concat` tests the concatenation of two strings
- `test_string_substring` tests the correctness of the substring extraction
- `test_string_index_of` ensures that the string returns the correct index of the specified symbol
- `test_string_equal_ignore_case` tests case-insensitive comparison of two strings
- `test_string_to_upper_case` tests conversion of the string to upper-case
- `test_string_replace` tests the correctness of the replacing operation

```cpp
#include <Arduino.h>
#include <unity.h>
```

(continues on next page)
String STR_TO_TEST;

void setUp(void) {
    // set stuff up here
    STR_TO_TEST = "Hello, world!";
}

void tearDown(void) {
    // clean stuff up here
    STR_TO_TEST = "";
}

void test_string_concat(void) {
    String hello = "Hello, ";
    String world = "world!";
    TEST_ASSERT_EQUAL_STRING(STR_TO_TEST.c_str(), (hello + world).c_str());
}

void test_string_substring(void) {
    TEST_ASSERT_EQUAL_STRING("Hello", STR_TO_TEST.substring(0, 5).c_str());
}

void test_string_index_of(void) {
    TEST_ASSERT_EQUAL(7, STR_TO_TEST.indexOf('w'));
}

void test_string_equal_ignore_case(void) {
    TEST_ASSERT_TRUE(STR_TO_TEST.equalsIgnoreCase("HELLO, WORLD"));
}

void test_string_to_upper_case(void) {
    STR_TO_TEST.toUpperCase();
    TEST_ASSERT_EQUAL_STRING("HELLO, WORLD!", STR_TO_TEST.c_str());
}

void test_string_replace(void) {
    STR_TO_TEST.replace('!', '?');
    TEST_ASSERT_EQUAL_STRING("Hello, world?", STR_TO_TEST.c_str());
}

void setup()
{
    delay(2000); // service delay
    UNITY_BEGIN();

    RUN_TEST(test_string_concat);
    RUN_TEST(test_string_substring);
    RUN_TEST(test_string_index_of);
    RUN_TEST(test_string_equal_ignore_case);
    RUN_TEST(test_string_to_upper_case);
    RUN_TEST(test_string_replace);

    UNITY_END(); // stop unit testing
}

void loop()
Now we are ready to upload tests to the board. To do this we can use the following:

- Test button on *PlatformIO Toolbar*,
- Test option in the *Project Tasks* menu, or
- **Tasks: Run Task... > PlatformIO Test** in the top menu:

After processing, we should see a detailed report about the testing results:
As we can see from the report, all our tests were successful!

**Adding Bluetooth LE features**

Now let's create a basic application that can interact with other BLE devices (e.g., phones). For example, the following code declares a BLE characteristic whose value can be printed to the serial port:

```cpp
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

#define SERVICE_UUID "4fac201-1fb5-459e-8fcc-c5c9c31914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

class MyCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic *pCharacteristic) {
    std::string value = pCharacteristic->getValue();
    if (value.length() > 0) {
      Serial.print("\nNew value: ");
      for (int i = 0; i < value.length(); i++)
        Serial.print(value[i]);
      Serial.println();
    }
  }
}
```

(continues on next page)
void setup() {
    Serial.begin(9600);
    BLEDevice::init("ESP32 BLE example");
    BLEServer *pServer = BLEDevice::createServer();
    BLEService *pService = pServer->createService(SERVICE_UUID);
    BLECharacteristic *pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_READ | 
        BLECharacteristic::PROPERTY_WRITE
    );
    pCharacteristic->setCallbacks(new MyCallbacks());
    pCharacteristic->setValue("Hello World");
    pService->start();
    BLEAdvertising *pAdvertising = pServer->getAdvertising();
    pAdvertising->start();
}

void loop() {
    delay(2000);
}

Now we can compile and upload this program to the board as described in the previous sections. To verify that our application works as expected, we can use any Android smartphone with the BLE feature and Nordic nRF Connect tool.

At first, we need to scan all advertising BLE devices and connect to the device called ESP32 BLE example. After successful connection to the board, we should see one “Unknown Service” with one “Unknown Characteristic” field:
To set the value, we need to send new text to the BLE characteristic:
The change of the value is printed to the serial monitor:
Conclusion

Now we have a project template for the ESP32-DevKitC board that we can use as boilerplate for later projects.

Get started with ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis

The goal of this tutorial is to demonstrate how simple it is to use VSCode to develop, run and debug a simple Wi-Fi project with the Espressif IoT Development Framework framework for the ESP32-DevKitC board.

- **Level:** Intermediate
- **Platforms:** Windows, Mac OS X, Linux

**Requirements:**

- Downloaded and installed VSCode
- Espressif ESP32 Dev Module
- An external debug adapter (e.g. Olimex ARM-USB-OCD)

**Contents**

- Setting Up the Project
• Adding Code to the Generated Project
• Debugging the Firmware
  – Setting Up the Hardware
• Writing Unit Tests
• Project Inspection
• Conclusion

Setting Up the Project

1. Click on “PlatformIO Home” button on the bottom PlatformIO Toolbar:

2. Click on “New Project”, select Espressif ESP32 Dev Module as the development board, Espressif IoT Development Framework as the framework and a path to the project location (or use the default one):
1. Create a new file `main.c` in `src_dir` folder and add the following code:

```c
/* WiFi softAP Example

This example code is in the Public Domain (or CC0 licensed, at your option.)

Unless required by applicable law or agreed to in writing, this software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
*/
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "lwip/err.h"
#include "lwip/sys.h"
#include "lwip/err.h"
#include "lwip/sys.h"
```

(continues on next page)
#define EXAMPLE_ESP_WIFI_SSID "mywifissid"
#define EXAMPLE_ESP_WIFI_PASS "mywifipass"
#define EXAMPLE_MAX_STA_CONN (3)

static const char *TAG = "wifi softAP";

static void wifi_event_handler(void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)
{
    if (event_id == WIFI_EVENT_AP_STACONNECTED) {
        wifi_event_ap_staconnected_t* event = (wifi_event_ap_staconnected_t*) event_data;
        ESP_LOGI(TAG, "station MACSTR join, AID=%d", MAC2STR(event->mac), event->aid);
    } else if (event_id == WIFI_EVENT_AP_STADISCONNECTED) {
        wifi_event_ap_stadisconnected_t* event = (wifi_event_ap_stadisconnected_t*) event_data;
        ESP_LOGI(TAG, "station MACSTR leave, AID=%d", MAC2STR(event->mac), event->aid);
    }
}

void wifi_init_softap()
{
    tcpip_adapter_init();
    ESP_ERROR_CHECK(esp_event_loop_create_default());

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &wifi_event_handler, NULL));

    wifi_config_t wifi_config = {
        .ap = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID),
            .password = EXAMPLE_ESP_WIFI_PASS,
            .max_connection = EXAMPLE_MAX_STA_CONN,
            .authmode = WIFI_AUTH_WPA_WPA2_PSK
        },
    };

    if (strlen(EXAMPLE_ESP_WIFI_PASS) == 0) {
        wifi_config.ap.authmode = WIFI_AUTH_OPEN;
    }

    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_AP, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());

    ESP_LOGI(TAG, "wifi_init_softap finished. SSID:%s password:%s", EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
}

void app_main()
//Initialize NVS
esp_err_t ret = nvs_flash_init();
if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
    ESP_ERROR_CHECK(nvs_flash_erase());
    ret = nvs_flash_init();
}
ESP_ERROR_CHECK(ret);
ESP_LOGI(TAG, "ESP_WIFI_MODE_AP");
wifi_init_softap();

**Warning:** Make sure this new file `main.c` is registered as source file using `idf_component_register` function in `src/CMakeLists.txt` file:

```c
idf_component_register(SRCS "main.c")
```

2. To compile the project use one of the following options:
   - Build option from the **Project Tasks** menu
   - Build button in **PlatformIO Toolbar**
   - **Task Menu** Tasks: Run Task... > **PlatformIO: Build** or in **PlatformIO Toolbar**
   - **Command Palette** View: Command Palette > **PlatformIO: Build**
   - **Hotkeys** cmd-alt-b / ctrl-alt-b:
3. If everything went well, we should see a successful result message in the terminal window:
4. To upload the firmware to the board we can use the following options:

- Upload option from the Project Tasks menu
- Upload button in PlatformIO Toolbar
- Command Palette View: Command Palette > PlatformIO: Upload
- Task Menu Tasks: Run Task... > PlatformIO: Upload
- Hotkeys cmd-alt-u / ctrl-alt-u:
5. Connect the board to your computer and update the default monitor speed to 115200 in platformio.ini file:

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = espidf
monitor_speed = 115200
```

6. Open Serial Monitor to observe the output from the board:
7. If everything went well, the board should be visible as a WiFi access point:
Debugging the Firmware

Setting Up the Hardware

In order to use *Debugging*, we need to connect an external JTAG probe and the board using the following pins:
1. Specify `debug_tool` in “platformio.ini” (Project Configuration File). In this tutorial, Olimex ARM-USB-OCD-H debug probe is used:

```ini
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = espidf
monitor_speed = 115200
debug_tool = olimex-arm-usb-ocd-h
```

2. To start the debug session we can use the following methods:

   - **Debug**: Start debugging in the top menu
   - **Start Debugging** option in the Quick Access menu
   - Hotkey button F5:
3. Walk through the code using control buttons, set breakpoints, and add variables to the Watch window:

![Debug View](image)

Writing Unit Tests

**Note:** Functions `setUp` and `tearDown` are used to initialize and finalize test conditions. Implementations of these functions are not required for running tests but if you need to initialize some variables before you run a test, you use the `setUp` function and if you need to clean up variables you use `tearDown` function.

For the sake of simplicity, let’s create a small library called `calculator`, implement several basic functions `addition`, `subtraction`, `multiplication`, `division` and test them using *Unit Testing* engine.

1. Create a new folder `calculator` in the `lib_dir` folder and add two new files `calculator.h` and `calculator.c` with the following contents:

```c
#include "calculator.h"

int addition (int a, int b);
```

(continues on next page)
int subtraction (int a, int b);
int multiplication (int a, int b);
int division (int a, int b);

#ifdef __cplusplus
}
#endif
#endif
// _CALCULATOR_H_

calculator.c:

#include "calculator.h"

int addition(int a, int b) {
    return a + b;
}

int subtraction(int a, int b) {
    return a - b;
}

int multiplication(int a, int b) {
    return a * b;
}

int division(int a, int b) {
    return a / b;
}

2. Create a new file test_calc.c to the folder test_dir and add basic tests for the calculator library:

#include <calculator.h>
#include <unity.h>

void test_function_calculator_addition(void) {
    TEST_ASSERT_EQUAL(32, addition(25, 7));
}

void test_function_calculator_subtraction(void) {
    TEST_ASSERT_EQUAL(20, subtraction(23, 3));
}

void test_function_calculator_multiplication(void) {
    TEST_ASSERT_EQUAL(50, multiplication(25, 2));
}

void test_function_calculator_division(void) {
    TEST_ASSERT_EQUAL(32, division(100, 3));
}

void main() {
    UNITY_BEGIN();

(continues on next page)
RUN_TEST(test_function_calculator_addition);
RUN_TEST(test_function_calculator_subtraction);
RUN_TEST(test_function_calculator_multiplication);
RUN_TEST(test_function_calculator_division);

UNITY_END();

3. Let’s run tests on the board and check the results. There should be a problem with test_function_calculator_division test:

4. Let’s fix the incorrect expected value and run tests again. After processing the results should be correct:
**Project Inspection**

For illustrative purposes, let’s imagine we need to find a function with the biggest memory footprint. Also, let’s introduce a bug to our project so *Static Code Analysis* can report it.

1. Open **PlatformIO Home** and navigate to **Inspect** section, select the current project and press **Inspect** button:
2. Project statistics:
3. The biggest function:
4. Possible bugs:
Conclusion

Now we have a project template for the ESP32-DevKitC board that we can use as boilerplate for later projects.

**STM32Cube HAL and Nucleo-F401RE: debugging and unit testing**

The goal of this tutorial is to demonstrate how simple it is to use PlatformIO IDE for Atom to develop, run and debug a basic blink project with STM32Cube framework for STM32 Nucleo-F401RE board.

- **Level:** Intermediate
- **Platforms:** Windows, Mac OS X, Linux

**Requirements:**

- Downloaded and installed PlatformIO IDE for Atom
- Install drivers for ST-LINK debug tool
- ST Nucleo F401RE development board

**Contents**

- Setting Up the Project
- Adding Code to the Generated Project
Setting Up the Project

At first step, we need to create a new project using PlatformIO Home Page (to open this page just press Home icon on the toolbar):

On the next step, we need to select ST Nucleo-F401RE as a development board, STM32Cube as a framework and a path to the project location (or use the default one):
Processing the selected project may take some amount of time (PlatformIO will download and install all required packages) and after these steps, we have a fully configured project that is ready for developing code with STM32Cube framework.

Adding Code to the Generated Project

Let’s add some actual code to the project. Firstly, we create two main files main.c and main.h in the src_dir folder. Right click on the src in the project window:
Add next content to `main.h`:

```c
#ifndef MAIN_H
#define MAIN_H

#include "stm32f4xx_hal.h"

#define LED_PIN GPIO_PIN_5
#define LED_GPIO_PORT GPIOA
#define LED_GPIO_CLK_ENABLE() __HAL_RCC_GPIOA_CLK_ENABLE()

#endif // MAIN_H
```

Add this code to `main.c`:

```c
#include "main.h"

void LED_Init();

int main(void) {
    HAL_Init();
    LED_Init();

    while (1)
    {
```

(continues on next page)
HAL_GPIO_TogglePin(LED_GPIO_PORT, LED_PIN);
HAL_Delay(1000);
}

void LED_Init() {
    LED_GPIO_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = LED_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    HAL_GPIO_Init(LED_GPIO_PORT, &GPIO_InitStruct);
}

void SysTick_Handler(void) {
    HAL_IncTick();
}

After this step, we created a basic blink project that is ready for compiling and uploading.

**Compiling and Uploading the Firmware**

Now we can build the project. To compile firmware we can use next options: Build option on the Project Tasks menu, Build button on PlatformIO Toolbar, using Command Palette View: Command Palette > PlatformIO: Build, using Task Menu Tasks: Run Task... > PlatformIO: Build or via hotkeys cmd-alt-b / ctrl-alt-b:
If everything went well, we should see the successful result in the terminal window:
To upload the firmware to the board we can use next options: Upload option on the Project Tasks menu, Upload button on PlatformIO Toolbar, using Command Palette View: Command Palette > PlatformIO: Upload, using Task Menu Tasks: Run Task... > PlatformIO: Upload or via hotkeys cmd-alt-u / ctrl-alt-u:
After successful uploading, the green LED2 should start blinking.

**Debugging the Firmware**

*Debugging* offers the easiest way to debug your board. To start debugging session you can use *Start debugging* option in PlatformIO Quick Access menu, *Debug: Start debugging* from the top menu or hotkey button F5:
We need to wait some time while PlatformIO is initializing debug session and when the first line after the main function is highlighted we are ready to debug:
We can walk through the code using control buttons, set breakpoints, see peripheral registers, add variables to **Watch** window:
Writing Unit Tests

Now let’s write some tests using Unit Testing feature that can help us test code directly on the target board. Unit Testing engine by default supports only three frameworks: Arduino, Espressif IoT Development Framework, Mbed. Since we decided to use STM32Cube we need to implement a custom test_transport to print testing results and specify that condition in “platformio.ini” (Project Configuration File):

```
[env:nucleo_f401re]
platform = ststm32
board = nucleo_f401re
framework = stm32cube
test_transport = custom
```

Also, we need to create a new folder test where the tests and custom test_transport implementation (described next) will be located:
We will use USART2 on ST Nucleo-F401RE board because it’s directly connected to the STLink debug interface and in OS it can be visible as a Virtual Com Port, so we don’t need any additional USB-UART converter. To implement the custom test_transport we need to create two files unittest_transport.h and unittest_transport.c and put them in the test_dir in the root folder of our project. In these files we need to implement the next four functions:

```c
void unittest_uart_begin();
void unittest_uart_putchar(char c);
void unittest_uart_flush();
void unittest_uart_end();
```

Implementation of unittest_transport.h:

```c
#ifndef UNITEST_TRANSPORT_H
#define UNITEST_TRANSPORT_H

#ifdef __cplusplus
extern "C" {
#endif

void unittest_uart_begin();
void unittest_uart_putchar(char c);
void unittest_uart_flush();
void unittest_uart_end();

#ifdef __cplusplus
}
#endif
#endif
```

(continues on next page)
Implementation of unittest_transport.c:

```
#include "unittest_transport.h"
#include "stm32f4xx_hal.h"

#define USARTx USART2
#define USARTx_CLK_ENABLE() __HAL_RCC_USART2_CLK_ENABLE()
#define USARTx_CLK_DISABLE() __HAL_RCC_USART2_CLK_DISABLE()
#define USARTx_RX_GPIO_CLK_ENABLE() __HAL_RCC_GPIOA_CLK_ENABLE()
#define USARTx_TX_GPIO_CLK_ENABLE() __HAL_RCC_GPIOA_CLK_ENABLE()
#define USARTx_RX_GPIO_CLK_DISABLE() __HAL_RCC_GPIOA_CLK_DISABLE()
#define USARTx_TX_GPIO_CLK_DISABLE() __HAL_RCC_GPIOA_CLK_DISABLE()
#define USARTx_FORCE_RESET() __HAL_RCC_USART2_FORCE_RESET()
#define USARTx_RELEASE_RESET() __HAL_RCC_USART2_RELEASE_RESET()

#define USARTx_TX_PIN GPIO_PIN_2
#define USARTx_TX_GPIO_PORT GPIOA
#define USARTx_TX_AF GPIO_AF7_USART2
#define USARTx_RX_PIN GPIO_PIN_3
#define USARTx_RX_GPIO_PORT GPIOA
#define USARTx_RX_AF GPIO_AF7_USART2

static UART_HandleTypeDef UartHandle;

void unittest_uart_begin()
{

  GPIO_InitTypeDef GPIO_InitStruct;

  USARTx_TX_GPIO_CLK_ENABLE();
  USARTx_RX_GPIO_CLK_ENABLE();
  USARTx_CLK_ENABLE();

  GPIO_InitStruct.Pin = USARTx_TX_PIN;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FAST;
  GPIO_InitStruct.Alternate = USARTx_TX_AF;
  HAL_GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = USARTx_RX_PIN;
  GPIO_InitStruct.Alternate = USARTx_RX_AF;
  HAL_GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStruct);

  UartHandle.Instance = USARTx;
  UartHandle.Init.BaudRate = 115200;
  UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
  UartHandle.Init.StopBits = UART_STOPBITS_1;

  HAL_UART_Init(&UartHandle);
}
```

(continues on next page)
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;

if (HAL_UART_Init(&UartHandle) != HAL_OK) {
    while(1){}
}

void unittest_uart_putchar(char c) {
    HAL_UART_Transmit(&UartHandle, (uint8_t*)(&c), 1, 1000);
}

void unittest_uart_flush(){}

void unittest_uart_end() {
    USARTx_CLK_DISABLE();
    USARTx_RX_GPIO_CLK_DISABLE();
    USARTx_TX_GPIO_CLK_DISABLE();
}

Now we need to add some test cases. Tests can be added to a single C file that may include multiple tests. First of all, we need to add three default functions: setUp, tearDown and main. setUp and tearDown are used to initialize and finalize test conditions. Implementations of these functions are not required for running tests but if you need to initialize some variables before you run a test, you use the setUp function and if you need to clean up variables you use tearDown function. In our example, we will use these functions to accordingly initialize and deinitialize LED.

main function acts as a simple program where we describe our test plan.

Let’s add a new file test_main.c to the folder test. Next basic tests for blinking routine will be implemented in this file:

• test_led_builtin_pin_number ensures that LED_PIN has the correct value
• test_led_state_high tests functions HAL_GPIO_WritePin and HAL_GPIO_ReadPin with GPIO_PIN_SET value
• test_led_state_low tests functions HAL_GPIO_WritePin and HAL_GPIO_ReadPin with GPIO_PIN_RESET value

Note:
• 2 sec delay is required since the board doesn’t support software resetting via Serial.DTR/RTS

#include "../src/main.h"
#include <unity.h>

void setUp(void) {
    LED_GPIO_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = LED_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
}
void tearDown(void) {
    HAL_GPIO_DeInit(LED_GPIO_PORT, LED_PIN);
}

void test_led_builtin_pin_number(void) {
    TEST_ASSERT_EQUAL(GPIO_PIN_5, LED_PIN);
}

void test_led_state_high(void) {
    HAL_GPIO_WritePin(LED_GPIO_PORT, LED_PIN, GPIO_PIN_SET);
    TEST_ASSERT_EQUAL(GPIO_PIN_SET, HAL_GPIO_ReadPin(LED_GPIO_PORT, LED_PIN));
}

void test_led_state_low(void) {
    HAL_GPIO_WritePin(LED_GPIO_PORT, LED_PIN, GPIO_PIN_RESET);
    TEST_ASSERT_EQUAL(GPIO_PIN_RESET, HAL_GPIO_ReadPin(LED_GPIO_PORT, LED_PIN));
}

int main() {
    HAL_Init(); // initialize the HAL library
    HAL_Delay(2000); // service delay
    UNITY_BEGIN();
    RUN_TEST(test_led_builtin_pin_number);

    for (unsigned int i = 0; i < 5; i++) {
        RUN_TEST(test_led_state_high);
        HAL_Delay(500);
        RUN_TEST(test_led_state_low);
        HAL_Delay(500);
    }

    UNITY_END(); // stop unit testing

    while(1){}
}

void SysTick_Handler(void) {
    HAL_IncTick();
}

Now we are ready to upload tests to the board. To do this we can use Test option from the Project Tasks menu, Tasks: Run Task... > PlatformIO Test option from the top menu or Test button on PlatformIO Toolbar:
After processing we should see a detailed report about the testing results:
Congratulations! As we can see from the report, all our tests went successfully!

**Conclusion**

Now we have a decent template that we can improve for our next more complex projects.

**Project Source Code**

The source code of this tutorial is available at https://github.com/platformio/platformio-examples/tree/develop/unit-testing/stm32cube

**Arduino and Nordic nRF52-DK: debugging and unit testing**

The goal of this tutorial is to demonstrate how simple it is to use VSCode to develop, run and debug a simple project with Arduino framework for Nordic nRF52-DK board.

- **Level**: Beginner
- **Platforms**: Windows, Mac OS X, Linux

**Requirements:**

- Downloaded and installed VSCode
• Install drivers for *J-LINK* debug tool
• *Nordic nRF52-DK* development board

## Contents

- Setting Up the Project
- Adding Code to the Generated Project
- Compiling and Uploading the Firmware
- Debugging the Firmware
- Writing Unit Tests
- Adding Bluetooth LE features
- Conclusion

### Setting Up the Project

At first step, we need to create a new project using PlatformIO Home Page (to open this page just press Home icon on the toolbar):
On the next step we need to select Nordic nRF52-DK as a development board, Arduino as a framework and a path to the project location (or use the default one):

Processing the selected project may take some amount of time (PlatformIO will download and install all required packages) and after these steps, we have a fully configured project that is ready for developing code with Arduino framework.

**Adding Code to the Generated Project**

Let’s add some actual code to the project. Firstly, we open a default main file main.cpp in the src_dir folder and replace its contents with the following:

```cpp
#include <Arduino.h>

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
    digitalWrite(LED_BUILTIN, LOW);
}
```

(continues on next page)
After this step, we created a basic blink project ready for compiling and uploading.

Compiling and Uploading the Firmware

Now we can build the project. To compile firmware we can use next options: Build option from the Project Tasks menu, Build button in PlatformIO Toolbar, Task Menu Tasks: Run Task... > PlatformIO: Build or in PlatformIO Toolbar, Command Palette View: Command Palette > PlatformIO: Build or via hotkeys cmd-alt-b / ctrl-alt-b:
If everything went well, we should see a successful result message in the terminal window:
To upload the firmware to the board we can use next options: Upload option from the Project Tasks menu, Upload button in PlatformIO Toolbar, Command Palette View: Command Palette > PlatformIO: Upload, using Task Menu Tasks: Run Task... > PlatformIO: Upload or via hotkeys cmd-alt-u / ctrl-alt-u:
After successful uploading, the green LED1 should start blinking.

**Debugging the Firmware**

*Debugging* offers the easiest way to debug the board. Firstly, we need to specify *debug_tool* in “platformio.ini” *(Project Configuration File)*. Since the board has an on-board JLink debug probe we can directly declare it in “platformio.ini” *(Project Configuration File):*

```ini
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk
framework = arduino
debug_tool = jlink
```

To start the debug session we can use next options: Debug: Start debugging from the top menu, Start Debugging option from Quick Access menu or hotkey button F5:
We need to wait some time while PlatformIO is initializing the debug session and when the first line after the main function is highlighted we are ready to debug:
We can walk through the code using control buttons, set breakpoints, add variables to Watch window:
Writing Unit Tests

Test cases can be added to a single file that may include multiple tests. First of all, in this file, we need to add four default functions: setUp, tearDown, setup and loop. Functions setUp and tearDown are used to initialize and finalize test conditions. Implementations of these functions are not required for running tests but if you need to initialize some variables before you run a test, you use the setUp function and if you need to clean up variables you use tearDown function. In our example we will use these functions to accordingly initialize and deinitialize LED. setup and loop functions act as a simple Arduino program where we describe our test plan.

Let's create test folder in the root of the project and add a new file test_main.cpp to this folder. Next basic tests for String class will be implemented in this file:

- test_string_concat tests the concatenation of two strings
- test_string_substring tests the correctness of the substring extraction
- test_string_index_of ensures that the string returns the correct index of the specified symbol
- test_string_equal_ignore_case tests case-insensitive comparison of two strings
- test_string_to_upper_case tests upper-case conversion of the string
- test_string_replace tests the correctness of the replacing operation

Note:
• 2 sec delay is required since the board doesn’t support software resetting via Serial.DTR/RTS

```cpp
#include <Arduino.h>
#include <unity.h>

String STR_TO_TEST;

void setUp(void) {
    // set stuff up here
    STR_TO_TEST = "Hello, world!";
}

void tearDown(void) {
    // clean stuff up here
    STR_TO_TEST = "";
}

void test_string_concat(void) {
    String hello = "Hello, ";
    String world = "world!";
    TEST_ASSERT_EQUAL_STRING(STR_TO_TEST.c_str(), (hello + world).c_str());
}

void test_string_substring(void) {
    TEST_ASSERT_EQUAL_STRING("Hello", STR_TO_TEST.substring(0, 5).c_str());
}

void test_string_index_of(void) {
    TEST_ASSERT_EQUAL(7, STR_TO_TEST.indexOf('w'));
}

void test_string_equal_ignore_case(void) {
    TEST_ASSERT_TRUE(STR_TO_TEST.equalsIgnoreCase("HELLO, WORLD!"));
}

void test_string_to_upper_case(void) {
    STR_TO_TEST.toUpperCase();
    TEST_ASSERT_EQUAL_STRING("HELLO, WORLD!", STR_TO_TEST.c_str());
}

void test_string_replace(void) {
    STR_TO_TEST.replace('!', '?');
    TEST_ASSERT_EQUAL_STRING("Hello, world!", STR_TO_TEST.c_str());
}

void setup() {
    delay(2000); // service delay
    unity_BEGIN();

    RUN_TEST(test_string_concat);
    RUN_TEST(test_string_substring);
    RUN_TEST(test_string_index_of);
    RUN_TEST(test_string_equal_ignore_case);
    RUN_TEST(test_string_to_upper_case);
    RUN_TEST(test_string_replace);
}
```
UNITY_END(); // stop unit testing

void loop()
{
}

Now we are ready to upload tests to the board. To do this we can use next options: Test button on PlatformIO Toolbar, Test option from the Project Tasks menu or Tasks: Run Task... > PlatformIO Test from the top menu:

After processing we should see a detailed report about the testing results:
As we can see from the report, all our tests were successful!

**Adding Bluetooth LE features**

To add the basic BLE functionality to our project we need to define the SoftDevice version and install a library called BLEPeripheral. Both these modifications can be specified in "platformio.ini" (Project Configuration File):

```
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk
framework = arduino
debug_tool = jlink
build_flags = -DNRF52_S132
lib_deps = BLEPeripheral
```

Now let’s create a basic application that can interact with other BLE devices (e.g. phone). For example, next code declares a BLE characteristic that controls the state of the LED1.

```
#include <Arduino.h>
#include <SPI.h>
#include <BLEPeripheral.h>
```
BLEPeripheral ledPeripheral = BLEPeripheral();
BLEService ledService = BLEService("19b10000e8f2537e4f6cd104768a1214");
BLECharCharacteristic ledCharacteristic = BLECharCharacteristic("19b10001e8f2537e4f6cd104768a1214", BLERead | BLEWrite);

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  ledPeripheral.setAdvertisedServiceUuid(ledService.uuid());
  ledPeripheral.addAttribute(ledService);
  ledPeripheral.addAttribute(ledCharacteristic);
  ledPeripheral.setLocalName("Nordic NRF52 DK");
  ledPeripheral.begin();
}

void loop()
{
  BLECentral central = ledPeripheral.central();
  if (central) {
    while (central.connected()) {
      if (ledCharacteristic.written()) {
        if (ledCharacteristic.value())
          digitalWrite(LED_BUILTIN, HIGH);
        else
          digitalWrite(LED_BUILTIN, LOW);
      }
    }
  }
}

Now we can compile and upload this program to the board as described in previous sections. To verify that our application works as expected, we can use any Android smartphone with BLE feature and Nordic nRF Connect tool.

At first, we need to scan all advertising BLE devices and connect to the device called Nordic NRF52 DK. After a successful connection to the board, we should see one “Unknown Service” with one “Unknown Characteristic” fields:
To switch the LED on or off we just need write 0 or 1 as UINT8 to the BLE characteristic:
Conclusion

Now we have a project template for Nordic nRF52-DK board that we can use as a boilerplate for the next projects.

**Zephyr and Nordic nRF52-DK: debugging, unit testing, project analysis**

The goal of this tutorial is to demonstrate how simple it is to use VSCode to develop, run and debug a simple Bluetooth project using Zephyr RTOS framework for the Nordic nRF52-DK board.

- **Level:** Intermediate
- **Platforms:** Windows, Mac OS X, Linux

**Requirements:**

- Downloaded and installed VSCode
- Install drivers for J-LINK debug tool
- Nordic nRF52-DK development board
Setting Up the Project

1. Click on “PlatformIO Home” button on the bottom PlatformIO Toolbar:

2. Click on “New Project”, select Nordic nRF52-DK as the development board, Zephyr RTOS as the framework and a path to the project location (or use the default one):
1. Create a new file `main.c` in `src_dir` folder and add the following code:

```c
// Copyright (c) 2015-2016 Intel Corporation
// SPDX-License-Identifier: Apache-2.0
#
#include <zephyr/types.h>
#include <stddef.h>
#include <sys/printk.h>
#include <sys/util.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#define DEVICE_NAME CONFIG_BT_DEVICE_NAME
#define DEVICE_NAME_LEN (sizeof(DEVICE_NAME) - 1)

// Set Advertisement data. Based on the Eddystone specification:
// https://github.com/google/eddystone/blob/master/protocol-specification.md
```

(continues on next page)
// https://github.com/google/eddystone/tree/master/eddystone-url

```c
static const struct bt_data ad[] = {
    BT_DATA_FLAGS(BT_LE_AD_NO_BREDR),
    BT_DATA_UUID16_ALL(0xaa, 0xfe),
    BT_DATA_SVC_DATA16(0xaa, 0xfe, 0x10, // Eddystone-URL frame type
                       0x00, // Calibrated Tx power at 0m
                       0x00, // URL Scheme Prefix http://www.
                     'z', 'e', 'p', 'h', 'y', 'r',
                     'p', 'i', 'o', 'j', 'e', 'c', 't',
                     0x08) // .org
};

// Set Scan Response data
static const struct bt_data sd[] = {
    BT_DATA_NAME_COMPLETE(DEVICE_NAME, DEVICE_NAME_LEN),
};

static void bt_ready(int err)
{
    if (err) {
        printk("Bluetooth init failed (err %d)\n", err);
        return;
    }

    printk("Bluetooth initialized\n");

    // Start advertising
    err = bt_le_adv_start(BT_LE_ADV_NCONN, ad, ARRAY_SIZE(ad),
                          sd, ARRAY_SIZE(sd));
    if (err) {
        printk("Advertising failed to start (err %d)\n", err);
        return;
    }

    printk("Beacon started\n");
}

void main(void)
{
    int err;

    printk("Starting Beacon Demo\n");

    // Initialize the Bluetooth Subsystem
    err = bt_enable(bt_ready);
    if (err) {
        printk("Bluetooth init failed (err %d)\n", err);
    }
}
```

2. By default Bluetooth feature is disabled, we can enable it by creating a new file `prj.conf` in `zephyr` folder and adding the following lines:
CONFIG_BT=y
CONFIG_BT_DEBUG_LOG=y
CONFIG_BT_DEVICE_NAME="Test beacon"

Compiling and Uploading the Firmware

1. To compile the project use one of the following options:
   - Build option from the Project Tasks menu
   - Build button in PlatformIO Toolbar
   - Task Menu Tasks: Run Task... > PlatformIO: Build or in PlatformIO Toolbar
   - Command Palette View: Command Palette > PlatformIO: Build
   - Hotkeys cmd-alt-b / ctrl-alt-b:

2. If everything went well, we should see a successful result message in the terminal window:
3. To upload the firmware to the board we can use the following options:

- Upload option from the **Project Tasks** menu
- Upload button in **PlatformIO Toolbar**
- **Command Palette** View: Command Palette > PlatformIO: Upload
- **Task Menu** Tasks: Run Task... > PlatformIO: Upload
- **Hotkeys** cmd-alt-u / ctrl-alt-u:
4. Connect the board to your computer and update the default monitor speed to 115200 in `platformio.ini` file:

   ```ini
   [env:hifive1-revb]
   platform = sifive
   board = hifive1-revb
   framework = zephyr
   monitor_speed = 115200
   ```

5. Open Serial Monitor to observe the output from the board:
6. If everything went well, the board should be visible as a beacon:
Debugging the Firmware

Since Nordic nRF52-DK includes an onboard debug probe we can use *Debugging* without any configuration.

1. To start a debug session we can use the following options:
   - Debug: Start debugging from the top menu
   - Start Debugging option from Quick Access menu
   - Hotkey button F5:
2. We can walk through the code using control buttons, set breakpoints, add variables to Watch window.
Writing Unit Tests

Note: Functions `setUp` and `tearDown` are used to initialize and finalize test conditions. Implementations of these functions are not required for running tests but if you need to initialize some variables before you run a test, you use the `setUp` function and if you need to clean up variables you use `tearDown` function.

For the sake of simplicity, let’s create a small library called `calculator`, implement several basic functions `add`, `sub`, `mul`, `div` and test them using `Unit Testing` engine.

1. PlatformIO uses a unit testing framework called `Unity`. `Unity` is not compatible with C library implemented in the framework. Let’s enable standard version of newlib C library in `prj.conf` file using the following config:

   ```
   CONFIG_NEWLIB_LIBC=y
   ```

2. Create a new folder `calculator` in the `lib` folder and add two new files `calculator.h` and `calculator.c` with the following contents:

   ```
   calculator.h:
   
   #ifndef _CALCULATOR_H_
   #define _CALCULATOR_H_
   ```
#ifdef __cplusplus
extern "C" {
#endif

int add (int a, int b);
int sub (int a, int b);
int mul (int a, int b);
int div (int a, int b);

#ifdef __cplusplus
}
#endif

// _CALCULATOR_H_

calculator.c:

#include "calculator.h"

int add(int a, int b) {
    return a + b;
}

int sub(int a, int b) {
    return a - b;
}

int mul(int a, int b) {
    return a * b;
}

3. Create a new file `test_calc.c to the folder test and add basic tests for calculator library:

#include <calculator.h>
#include <unity.h>

void test_function_calculator_addition(void) {
    TEST_ASSERT_EQUAL(32, add(25, 7));
}

void test_function_calculator_subtraction(void) {
    TEST_ASSERT_EQUAL(20, sub(23, 3));
}

void test_function_calculator_multiplication(void) {
    TEST_ASSERT_EQUAL(50, mul(25, 2));
}

void test_function_calculator_division(void) {
    TEST_ASSERT_EQUAL(32, div(100, 3));
}

void main() {
    UNITY_BEGIN();

(continues on next page)
RUN_TEST(test_function_calculator_addition);
RUN_TEST(test_function_calculator_subtraction);
RUN_TEST(test_function_calculator_multiplication);
RUN_TEST(test_function_calculator_division);

UNITY_END();

4. Let’s run tests on the board and check the results. There should be a problem with `test_function_calculator_division` test:

![Screenshot of test results](image)

5. Let’s fix the incorrect expected value, run tests again. After processing the results should be correct:
Project Inspection

For illustrative purposes, let’s imagine we need to find a function with the biggest memory footprint. Also, let’s introduce a bug to our project so Static Code Analysis can report it.

1. **Open PlatformIO Home and navigate to Inspect section**, select the current project and press **Inspect** button:
2. Project statistics:
3. The biggest function:
4. Possible bugs:
Conclusion

Now we have a project template for Nordic Nordic nRF52-DK board that we can use as a boilerplate for the next projects.

RISC-V ASM Video Tutorial

An introduction to using SiFive and Assembly language on the SiFive HiFive1 by Martin Fink, Chief Technology Officer at Western Digital.

Source Files

A demo source code is published on Github: https://github.com/martin-robert-fink/superBlink.git

It is already pre-configured PlatformIO project:

- Clone it or download
- Open in VSCode
- Happy coding and debugging!
Video Collection

- Part 1 of 12 | Introduction
- Part 2 of 12 | Setting Up
- Part 3 of 12 | Tour PlatformIO
- Part 4 of 12 | C Code Wrapper
- Part 5 of 12 | HiFive Docs
- Part 6 of 12 | Understanding GPIO
- Part 7 of 12 | setupGPIO
- Part 8 of 12 | Debug setupGPIO
- Part 9 of 12 | setLED
- Part 10 of 12 | Debug setLED
- Part 11 of 12 | Delay
- Part 12 of 12 | Final and Conclusion

1.5.2 Project Examples

Pre-configured projects with source code are located in PlatformIO Examples repository.

1.5.3 Community Projects

- PlatformIO DIY Projects & Tutorials at Hackster.io
1.5.4 Community Video Tutorials

- PlatformIO Video Collection on YouTube
- Next-generation IDE for your RISC-V Product in 20 Minutes by CEO of PlatformIO
- Use the PlatformIO Debugger on the ESP32 Using an ESP-prog
- RISC-V ASM Tutorial
- PlatformIO for Arduino, ESP8266, and ESP32 Tutorial
- Free Inline Debugging for ESP32 and Arduino Sketches
- PlatformIO, Arduino IDE
- ESP32 PlatformIO
- A Better Arduino IDE - Getting Started with PlatformIO
- PlatformIO - Using External Libraries

1.6 “platformio.ini” (Project Configuration File)

Each PlatformIO project has a configuration file named platformio.ini in the root directory for the project. This is a INI-style file.

platformio.ini has sections (each denoted by a [header]) and key / value pairs within the sections. Lines beginning with ; are ignored and may be used to provide comments.

Multiple value options can be specified in two ways:

1. Split values with “,” (comma + space)
2. Multi-line format, where each new line starts with at least two spaces

There are two required sections:

- **PlatformIO Core (CLI) settings:** Section [platformio]
- **Environment settings:** Section [env]

The other sections are optional to include. Here are the allowed sections and their allowed contents:

1.6.1 Section [platformio]

- **Generic options**
  - description
  - default_envs
  - extra_configs
- **Directory options**
  - core_dir
  - globallib_dir
  - platforms_dir
The platform.ini platformio section is used for overriding the default configuration options for PlatformIO Core (CLI).

**Note:** Relative path is allowed for directory option:

- `~` will be expanded to user’s home directory
- `../` or `..\` go up to one folder

There is a `$PROJECT_HASH` template variable. You can use it in a directory path. It will be replaced by a SHA1[0:10] hash of the full project path. This is very useful to declare a global storage for project workspaces. For example, `/tmp/pio-workspaces/$PROJECT_HASH` (Unix) or `${sysenv.TEMP}/pio-workspaces/$PROJECT_HASH` (Windows). You can set a global workspace directory using the system environment variable `PLATFORMIO_WORKSPACE_DIR`.

See the available directory ***_dir options below.

### Generic options

#### description

Type: String | Multiple: No  
Short description of the project. PlatformIO uses it for PlatformIO Home in the multiple places.

#### default_envs

Type: String | Multiple: Yes  
The `pio run` command processes all environments `[env:***]` by default if the `pio run --environment` option is not specified. `default_envs` allows one to define which environments that should be processed by default.

Also, `Debugging` checks this option when looking for debug environment.
This option can also be configured by the global environment variable `PLATFORMIO_DEFAULT_ENVS`.

Example:

```
[platformio]
default_envs = uno, nodemcu

[env:uno]
platform = atmelavr
framework = arduino
board = uno

[env:nodemcu]
platform = espressif8266
framework = arduino
board = nodemcu

[env:teensy31]
platform = teensy
framework = arduino
board = teensy31

[env:lpmsp430g2553]
platform = timsp430
framework = arduino
board = lpmsp430g2553
build_flags = -D LED_BUILTIN=RED_LED
```

**extra_configs**

Type: String (Pattern) | Multiple: Yes

This option allows extending a base “`platformio.ini` (Project Configuration File)” with extra configuration files. The format and rules are the same as for the “`platformio.ini` (Project Configuration File). A name of the configuration file can be any.

`extra_configs` can be a single path to an extra configuration file or a list of them. Please note that you can use Unix shell-style wildcards:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches everything</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
<tr>
<td>[seq]</td>
<td>matches any character in seq</td>
</tr>
<tr>
<td>![seq]</td>
<td>matches any character not in seq</td>
</tr>
</tbody>
</table>

**Note:** If you declare the same pair of “group” + “option” in an extra configuration file which was previously declared in a base “`platformio.ini` (Project Configuration File), it will be overwritten with a value from extra configuration.

**Example**

*Base “`platformio.ini`”*

```
[platformio]
extra_configs =
```
extra_envs.ini
extra_debug.ini

; Global data for all [env:***]
[env]
    platform = espressif32
    framework = espidf

; Custom data group
; can be use in [env:***] via ${common.***}
[common]
    debug_flags = -D RELEASE
    lib_flags = -lc -lm

[env:esp-wrover-kit]
    board = esp-wrover-kit
    build_flags = ${common.debug_flags}

"extra_envs.ini"

[env:esp32dev]
    board = esp32dev
    build_flags = ${common.lib_flags} ${common.debug_flags}

[env:lolin32]
    platform = espressif32
    framework = espidf
    board = lolin32
    build_flags = ${common.debug_flags}

"extra_debug.ini"

# Override base "common.debug_flags"
[common]
    debug_flags = -D DEBUG=1

[env:lolin32]
    build_flags = -Og

After a parsing process, configuration state will be the next:

[common]
    debug_flags = -D DEBUG=1
    lib_flags = -lc -lm

[env:esp-wrover-kit]
    platform = espressif32
    framework = espidf
    board = esp-wrover-kit
    build_flags = ${common.debug_flags}

[env:esp32dev]
    platform = espressif32
    framework = espidf
    board = esp32dev
    build_flags = ${common.lib_flags} ${common.debug_flags}
Directory options

core_dir

Type: DirPath | Multiple: No

The core_dir variable points out the directory used for all development platform packages (toolchains, frameworks, SDKs, upload and debug tools), global libraries for Library Dependency Finder (LDF), and other PlatformIO Core service data. The size of this folder will depend on the number of installed development platforms.

The default value is the user’s home directory:
- Unix ~/.platformio
- Windows %HOMEPATH%\platformio

This option can also be configured by the global environment variable PLATFORMIO_CORE_DIR.

Example:

```
[platformio]
core_dir = /path/to/custom/pio-core/storage
```

globallib_dir

Type: DirPath | Multiple: No | Default: “core_dir/lib”

Global library storage for PlatfrmIO projects and Library Management where Library Dependency Finder (LDF) looks for dependencies.

This option can also be configured by the global environment variable PLATFORMIO_GLOBALLIB_DIR.

platforms_dir

Type: DirPath | Multiple: No | Default: “core_dir/platforms”

Global storage where PlatformIO Package Manager installs Development Platforms.

This option can also be configured by the global environment variable PLATFORMIO_PLATFORMS_DIR.

packages_dir

Type: DirPath | Multiple: No | Default: “core_dir/packages”

Global storage where PlatformIO Package Manager installs Development Platforms dependencies (toolchains, Frameworks, SDKs, upload and debug tools).

This option can also be configured by the global environment variable PLATFORMIO_PACKAGES_DIR.
**cache_dir**

Type: DirPath | Multiple: No | Default: "core_dir/cache"

*PlatformIO Core (CLI)* uses this folder to store caching information (requests to PlatformIO Registry, downloaded packages and other service information).

To reset a cache, please run *pio update* command.

This option can also be configured by the global environment variable `PLATFORMIO_CACHE_DIR`.

**build_cache_dir**

Type: DirPath | Multiple: No | Default: None (Disabled)

*PlatformIO Core (CLI)* uses this folder to store derived files from a build system (objects, firmwares, ELFs). These files are shared between all build environments. To speed up a build process, you can use the same cache folder between different projects if they depend on the same development platform and framework.

This option can also be configured by the global environment variable `PLATFORMIO_BUILD_CACHE_DIR`.

The example of "platformio.ini" (Project Configuration File) below instructs PlatformIO Build System to check `build_cache_dir` for already compiled objects for *STM32Cube* and project source files. The cached object will not be used if the original source file was modified or build environment has a different configuration (new build flags, etc):

```
[platformio]
; Set a path to a cache folder
build_cache_dir =

; Examples:
; (Unix) build_cache_dir = /path/to/cache/folder
; (Windows) build_cache_dir = C:/path/to/cache/folder

[env:bluepill_f103c6]
platform = ststm32
framework = stm32cube
board = bluepill_f103c6

[env:nucleo_f411re]
platform = ststm32
framework = stm32cube
board = nucleo_f411re
```

**workspace_dir**

Type: DirPath | Multiple: No | Default: “Project/.pio”

The path to a project workspace directory where PlatformIO keeps by default compiled objects, static libraries, firmwares, and external library dependencies. It is used by these options:

- `build_dir`
- `libdeps_dir`

The default value is `.pio` and means that folder is located in the root of project.

This option can also be configured by the global environment variable `PLATFORMIO_WORKSPACE_DIR`. 
**build_dir**

**Warning:** PLEASE DO NOT EDIT FILES IN THIS FOLDER. PlatformIO will overwrite your changes on the next build. **THIS IS A CACHE DIRECTORY.**

Type: DirPath | Multiple: No | Default: “workspace_dir/build”

*PlatformIO Build System* uses this folder for project environments to store compiled object files, static libraries, firmwares and other cached information. It allows PlatformIO to build source code extremely fast!

You can delete this folder without any risk! If you modify “platformio.ini” (*Project Configuration File*), then PlatformIO will remove this folder automatically. It will be created on the next build operation.

This option can also be configured by the global environment variable `PLATFORMIO_BUILD_DIR`.

**Note:** If you have any problems with building your project environments which are defined in “platformio.ini” (*Project Configuration File*), then TRY TO DELETE this folder. In this situation you will remove all cached files without any risk. Also, you can use “clean” target for *pio run --target* command.

**libdeps_dir**

Type: DirPath | Multiple: No | Default: “workspace_dir/libdeps”

Internal storage where *Library Management* will install project dependencies (*lib_deps*).

This option can also be configured by the global environment variable `PLATFORMIO_LIBDEPS_DIR`.

**include_dir**

Type: DirPath | Multiple: No | Default: “Project/include”

The path to project’s default header files. PlatformIO uses it for the *pio run* command. The default value is include meaning an *include* directory located under the root directory of the project. This path will be added to CPPPATH of the build environment.

If you need to add extra include directories to CPPPATH scope, please use `build_flags` with `-I /path/to/extra/dir` option.

This option can also be configured by the global environment variable `PLATFORMIO_INCLUDE_DIR`.

**src_dir**

Type: DirPath | Multiple: No | Default: “Project/src”

The path to the project’s directory with source code. PlatformIO uses it for the *pio run* command. The default value is *src* meaning a *src* directory located in the root directory of the project.

This option can also be configured by the global environment variable `PLATFORMIO_SRC_DIR`.

**Note:** This option is useful for people who migrate from Arduino IDE where the source directory should have the same name as the main source file. See example project with own source directory.
**lib_dir**

Type: DirPath | Multiple: No | Default: “Project/lib”

You can put your own/private libraries here. The source code of each library should be placed in separate directory, like `lib/private_lib/ [here are source files]`. This directory has the highest priority for *Library Dependency Finder (LDF)*.

The default value is `lib`, meaning a `lib` directory located in the root of the project.

This option can also be configured by the global environment variable `PLATFORMIO_LIB_DIR`.

For example, see how the `Foo` and `Bar` libraries are organized:

```
|--lib
  | |--Bar
  |   | |--docs
  |   | |--examples
  |   | |--src
  |   |   |--Bar.c
  |   |   |--Bar.h
  |   |--Foo
  |     | |-- Foo.c
  |     | |-- Foo.h
  |     |-- platformio.ini
 |--src
     |-- main.c
```

Then in `src/main.c` you should use:

```c
#include <Foo.h>
#include <Bar.h>

// rest of H/C/CPP code
```

PlatformIO will find your libraries automatically, configure the preprocessor’s include paths and build them.

**data_dir**

Type: DirPath | Multiple: No | Default: “Project/data”

Data directory to store contents and *Using Filesystem*. The default value is `data` that means that folder is located in the root of the project.

This option can also be configured by the global environment variable `PLATFORMIO_DATA_DIR`.

**test_dir**

Type: DirPath | Multiple: No | Default: “Project/test”

The directory where *Unit Testing* engine will look for the tests. The default value is `test`, meaning a `test` directory located in the root of the project.

This option can also be configured by the global environment variable `PLATFORMIO_TEST_DIR`. 
boards_dir

Type: DirPath | Multiple: No | Default: “Project/boards”

The location of project-specific board definitions. Each project may choose a suitable directory name. The default value is boards, meaning a “boards” directory located in the root of the project.

By default, PlatformIO looks for boards in this order:

1. Project boards_dir (as defined by this setting)
2. Global core_dir/boards
3. Development platform core_dir/platforms/*/boards.

This option can also be configured by the global environment variable PLATFORMIO_BOARDS_DIR.

shared_dir

Type: DirPath | Multiple: No | Default: “Project/shared”

Remote Development uses this folder to synchronize extra files between remote machine. For example, you can share extra_scripts.

Please note that these folders are automatically shared between remote machine with pio remote run --force-remote or pio remote test --force-remote commands:

- lib_dir
- include_dir
- src_dir
- boards_dir
- data_dir
- test_dir

The default value is shared, meaning a directory named “shared” located in the root of the project.

This option can also be configured by the global environment variable PLATFORMIO_SHARED_DIR.

1.6.2 Section [env]

- Common [env]
- Environment [env:NAME]
- Options

Each project may have multiple configuration environments defining the available project tasks for building, programming, debugging, unit testing, device monitoring, library dependencies, etc. The configuration environments are declared using [env] sections in “platformio.ini” (Project Configuration File).

The allowed options are listed under Options.
Common [env]

An optional configuration environment with common options that will be shared between all [env:NAME] environments in the platform.ini file. It is very useful if the configuration file has a lot of environments [env:NAME] and they share common settings.

For example:

```
[env]
platform = ststm32
framework = stm32cube
board = nucleo_l152re
lib_deps = Dep1, Dep2

[env:release]
build_flags = -D RELEASE
lib_deps = ${env.lib_deps}
        Dep3

[env:debug]
build_type = debug
build_flags = -D DEBUG
lib_deps = DepCustom
```

In this example we have two configuration environments release and debug. This is equivalent to duplicating all options as shown below:

```
[env:release]
platform = ststm32
framework = stm32cube
board = nucleo_l152re
build_flags = -D RELEASE
lib_deps = Dep1, Dep2, Dep3

[env:debug]
platform = ststm32
framework = stm32cube
board = nucleo_l152re
build_type = debug
build_flags = -D DEBUG
lib_deps = DepCustom
```

Environment [env:NAME]

A section with an env: prefix defines a working environment for pio run, pio test, pio check, pio debug and other commands. Multiple [env:NAME] environments with different NAME are allowed. Every project must define at least one working environment.

Each environment must have a unique NAME. The valid chars for NAME are letters a-z, numbers 0-9, special char _ (underscore). For example, [env:hello_world].

If you have multiple working environments and you need to process only a few of them, the commands mentioned above accept the -e, --environment option to select a subset of the working environments to process. The [platformio] default_envs option can be used to define a default set of working environments for the commands to process.
Options

Platform options

- platform
- platform_packages
- framework
- board
- board_build.mcu
- board_build.f_cpu
- board_build.ldscript
- More options

platform

Type: String | Multiple: No

Development Platforms name.

PlatformIO allows one to use specific version of platform using Semantic Versioning (X.Y.Z=MAJOR.MINOR.PATCH) or VCS (Git, Mercurial and Subversion).

Version specifications can take any of the following forms:

- 1.2.3: an exact version number. Use only this exact version
- ^1.2.3: any compatible version (exact version for 1.x.x versions)
- ~1.2.3: any version with the same major and minor versions, and an equal or greater patch version
- >1.2.3: any version greater than 1.2.3. >=, <=, and <= are also possible
- >0.1.0, !=0.2.0, <0.3.0: any version greater than 0.1.0, not equal to 0.2.0 and less than 0.3.0

Examples:

[env:the_latest_version]
platform = atmelavr

[env:exact_version]
platform = atmelavr@1.2.3

[env:specific_major_version]
platform = atmelavr@^1.2.3

[env:specific_major_and_minor_version]
platform = atmelavr@~1.2.3

[env:development_version_by_git]
platform = https://github.com/platformio/platform-ststm32.git

(continues on next page)
platform_packages

Type: String | Multiple: Yes

Configure custom packages per a build environment. You can also override default packages by Development Platforms using the same name. Packages will be installed in packages_dir.

Examples:

```python
[env:override_default_toolchain]
platform = atmelavr
platform_packages =
  ; use GCC AVR 5.0+
  toolchain-gccarmnoneeabi@1.50000.0

[env:override_framework]
platform = espressif8266
platform_packages =
  ; use upstream Git version
  framework-arduinoespressif8266 @ https://github.com/esp8266/Arduino.git

[env:external_package]
platform = ststm32
platform_packages =
  ; latest openOCD from PlatformIO Package Registry
  tool-openocd
  ; source code of ST-Link
  tool-stlink-source @ https://github.com/texane/stlink.git
```

framework

Type: String | Multiple: Yes

Frameworks name.

board

Type: String (ID) | Multiple: No

PlatformIO has pre-configured settings for the most popular boards:

- build configuration
- upload configuration
- debugging configuration
**connectivity information, etc.**

You can find a valid `board` ID in *Boards* catalog, *Boards Explorer* or *pio boards* command.

**board_build.mcu**

**Type: String | Multiple: No**

`board_build.mcu` is a microcontroller(MCU) type that is used by compiler to recognize MCU architecture. The correct type of `board_build.mcu` depends on platform library. For example, the list of `board_build.mcu` for “megaAVR Devices” is described here.

The full list of `board_build.mcu` for the popular embedded platforms you can find in *Boards* section of *Development Platforms*. See “Microcontroller” column.

**board_build.f_cpu**

**Type: Integer | Multiple: No**

The option `board_build.f_cpu` is used to define MCU frequency (Hertz, Clock). A format of this option is C-like long integer value with `L` suffix. The 1 Hertz is equal to `1L`, then 16 MHz (Mega Hertz) is equal to `16000000L`.

The full list of `board_build.f_cpu` for the popular embedded platforms you can find in *Boards* section of *Development Platforms*. See “Frequency” column. You can overclock a board by specifying a `board_build.f_cpu` value other than the default.

**board_build.ldscript**

**Type: String | Multiple: No**

Path to the linker script to be used instead of the one defined by a framework. This is useful for specifying a modified linker script, for example, when an application requires a special memory section for a bootloader.

**More options**

You can override any board option declared in manifest file using the next format `board_{OBJECT.PATH}`, where `{OBJECT.PATH}` is an object path in JSON manifest. Please navigate to “boards” folder of PlatformIO development platforms and open JSON file to list all available options.

For example, Manifest: Espressif ESP32 Dev Module:

```
[env:custom_board_options]
; Custom CPU Frequency
board_build.f_cpu = 160000000L

; Custom FLASH Frequency
board_build.f_flash = 80000000L

; Custom FLASH Mode
board_build.flash_mode = qio

; Custom linker script
board_build.ldscript = /path/to/ldscript.ld
```

(continues on next page)
Build options

- **build_type**
- **build_flags**
  - *Built-in Variables*
  - *Dynamic build flags*
- **src_build_flags**
- **build_unflags**
- **src_filter**
- **targets**

**build_type**

Type: String | Multiple: No | Default: release

See extended documentation for *Build Configurations*.

**build_flags**

Type: String | Multiple: Yes

These flags/options affect the preprocessing, compilation, assembly and linking processes for C and C++ code. All compiler and linker flags can be used. Here is a list of some common options.

In spite of the name, **CPPDEFINES** rows also applies to the C compiler.
### Format

<table>
<thead>
<tr>
<th>Format</th>
<th>Affects build variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-D name</code></td>
<td>CPPDE-FINES</td>
<td>Predefine <code>name</code> as a macro, with definition 1.</td>
</tr>
<tr>
<td><code>-D name=definition</code></td>
<td>CPPDE-FINES</td>
<td>The contents of <code>definition</code> are tokenized and processed as if they appeared during translation phase three in a <code>#define</code> directive.</td>
</tr>
<tr>
<td><code>-U name</code></td>
<td>CPPDE-FINES</td>
<td>Cancel any previous definition of <code>name</code>, either built in or provided with a <code>-D</code> option.</td>
</tr>
<tr>
<td><code>-Wp,option</code></td>
<td>CPPFLAGS</td>
<td>Bypass the compiler driver and pass <code>option</code> directly through to the preprocessor.</td>
</tr>
<tr>
<td><code>-Wall</code></td>
<td>CCFLAGS</td>
<td>Turn on all optional warnings which are desirable for normal code.</td>
</tr>
<tr>
<td><code>-Werror</code></td>
<td>CCFLAGS</td>
<td>Make all warnings into hard errors. With this option, if any source code triggers warnings, the compilation will be aborted.</td>
</tr>
<tr>
<td><code>-w</code></td>
<td>CCFLAGS</td>
<td>Suppress all warnings, including those which GNU CPP issues by default.</td>
</tr>
<tr>
<td><code>-include file</code></td>
<td>CCFLAGS</td>
<td>Process <code>file</code> as if <code>#include &quot;file&quot;</code> appeared as the first line of the primary source file.</td>
</tr>
<tr>
<td><code>-Idir</code></td>
<td>CPPPATH</td>
<td>Add the directory <code>dir</code> to the list of directories to be searched for header files.</td>
</tr>
<tr>
<td><code>-Wa,option</code></td>
<td>ASFLAGS, CCFLAGS</td>
<td>Pass <code>option</code> as an option to the assembler. If <code>option</code> contains commas, it is split into multiple options at the commas.</td>
</tr>
<tr>
<td><code>-Wl,option</code></td>
<td>LINKFLAGS</td>
<td>Pass <code>option</code> as an option to the linker. If <code>option</code> contains commas, it is split into multiple options at the comma.</td>
</tr>
<tr>
<td><code>-llibrary</code></td>
<td>LIBS</td>
<td>Search the <code>library</code> named library when linking</td>
</tr>
<tr>
<td><code>-Ldir</code></td>
<td>LIBPATH</td>
<td>Add directory <code>dir</code> to the list of directories to be searched for <code>-l</code>.</td>
</tr>
</tbody>
</table>

This option can also be set by global environment variable `PLATFORMIO_BUILD_FLAGS`.

For more detailed information about available flags/options go to:

- Options to Request or Suppress Warnings
- Options for Debugging Your Program
- Options That Control Optimization
- Options Controlling the Preprocessor
- Passing Options to the Assembler
- Options for Linking
- Options for Directory Search

Examples:

```yaml
[env:specific_defines]
build_flags =
  -DFOO -DBAR=1
  -D BUILD_ENV_NAME=$PIOENV
  -D CURRENT_TIME=$UNIX_TIME
  -DFLOAT_VALUE=1.23457e+07

[env:string_defines]
build_flags =
  -DHELLO="World!"
  '-DWIFI_PASS="My password"
  ; Password with special chars: My pass\'word
  -DWIFI_PASS="\My\ pass\'word\"
```

(continues on next page)
[env:specific_inclibs]
build_flags =
  -I/opt/include
  -L/opt/lib
  -lfoo

[env:ignore_incremental_builds]
; We dynamically change the value of "LAST_BUILD_TIME" macro,
; PlatformIO will not cache objects
build_flags = -DLAST_BUILD_TIME=$UNIX_TIME

Note: If you need to control build flags that are specific for debug configuration please refer to debug_build_flags.

Built-in Variables

You can inject the built-in variables into your build flags, such as:

- $PYTHONEXE, full path to current Python interpreter
- $UNIX_TIME, current time in Unix format
- $PIOENV, name of build environment from “platformio.ini” (Project Configuration File)
- $PIOPLATFORM, name of development platform
- $PIOFRAMEWORK, a list of frameworks
- $PROJECT_DIR, project directory
- $PROJECT_CORE_DIR, PlatformIO Core directory, see core_dir
- $PROJECT_BUILD_DIR, project build directory per all environments
- $BUILD_DIR, build directory per current environment

See the full list of PlatformIO variables.

Please use target envdump for the pio run --target command to see ALL variable values for a build environment.

Dynamic build flags

PlatformIO allows users to run an external command/script which outputs build flags into STDOUT by prepending the shell command with a ! character. PlatformIO will automatically replace commands with their output when appending flags to build environments.

You can use any shell or programming language.

This external command will be called on each pio run command before building/uploading process.

Use cases:

- Macro with the latest VCS revision/tag “on-the-fly”
- Generate dynamic headers (*.h)
- Process media content before generating SPIFFS image
- Make some changes to source code or related libraries
Note: If you need more advanced control and would like to apply changes to a PlatformIO Build System environment, please refer to Advanced Scripting.

Example:

```ini
[env:generate_flags_with_external_command]
build_flags = !cmd_or_path_to_script

; Unix only, get output from internal command
build_flags = !echo "-DSOME_MACRO=$(some_cmd arg1 --option1)"
```

Use Case: Create a “PIO_SRC_REV” macro with the latest Git revision

This example includes a separate file named `git_rev_macro.py`, to be placed in the same directory as `platformio.ini`.

`platformio.ini`:

```ini
[env:git_revision_macro]
build_flags = !python git_rev_macro.py
```

`git_rev_macro.py`:

```python
import subprocess

revision = 
    subprocess.check_output(['git', 'rev-parse', 'HEAD'])
    .strip()
    .decode('utf-8')

print("-DGIT_REV='\"%s\'\"" % revision)
```

**src_build_flags**

Type: String | Multiple: Yes

An option `src_build_flags` has the same behavior as `build_flags` but will be applied only for project source files in the `src_dir` directory.

This option can also be set by the global environment variable `PLATFORMIO_SRC_BUILD_FLAGS`.

**build_unflags**

Type: String | Multiple: Yes

Selectively remove base/initial flags that were set by the development platform.

```ini
[env:unflags]
build_unflags = -Os -std=gnu++11
build_flags = -O2
```

**src_filter**

Type: String (Templates) | Multiple: Yes
This option allows one to specify which source files should be included or excluded from `src_dir` for a build process. Filter supports two templates:

- `+<PATH>` include template
- `-<PATH>` exclude template

`PATH` is relative to `src_dir`. All patterns will be applied in their order of definition. GLOB Patterns are allowed.

By default, `src_filter` is predefined to `+<*> -<.git/> -<.svn/> -<example/> -<examples/> -<test/> -<tests/>`, meaning “include ALL files, then exclude the .git and svn repository folders and the example... folder.

This option can also be set by the global environment variable `PLATFORMIO_SRC_FILTER`.

### targets

**Type:** String | **Multiple:** Yes

A list of targets which will be processed by the `pio run` command by default. You can enter more than one target, if separated by comma+space “,”.

Please follow to `pio run --list-targets` documentation for the other targets.

**Examples**

1. Build a project using Release Configuration, upload the firmware, and start Serial Monitor automatically:

   ```
   [env:upload_and_monitor]
   targets = upload, monitor
   ```

2. Build a project using Debug Configuration.

**Tip!** You can use these targets like an option to `pio run --target` command. For example:

   ```
   # clean project
   pio run -t clean
   # dump current build environment
   pio run --target envdump
   ```

When no targets are defined, PlatformIO will build only sources by default.

### Library options

**See also:**

Please make sure to read Library Dependency Finder (LDF) guide first.

- `lib_deps`
- `lib_ignore`
- `lib_extra_dirs`
- `lib_ldf_mode`
- `lib_compat_mode`
lib_deps

See also:
Please make sure to read Library Dependency Finder (LDF) guide first.

Type: String | Multiple: Yes

Specify project dependencies that should be installed automatically to `libdeps_dir` before environment processing.

If you have multiple build environments that depend on the same libraries, you can use Dynamic variables to use common configuration.

Valid forms

```
[env:myenv]
lib_deps = LIBRARY_1, LIBRARY_2, LIBRARY_N

[env:myenv2]
lib_deps =
  LIBRARY_1
  LIBRARY_2
  LIBRARY_N
```

The each line with `LIBRARY_1`... `LIBRARY_N` will be passed automatically to `pio lib install` command.

Please check `pio lib install` for the valid declaration formats.

Example:

```
[env:myenv]
lib_deps =

; name-based (built-in library in framework)
SPI

; owner-based declaration
knolleary/PubSubClient

; SemVer specification
bblanchon/ArduinoJson @ ^5.6,!=5.4

; external Git resource
https://github.com/gioblu/PJON.git#v2.0

; custom name
IRremoteESP8266=https://github.com/markszabo/IRremoteESP8266/archive/master.zip
```

lib_ignore

See also:
Please make sure to read Library Dependency Finder (LDF) guide first.

Type: String | Multiple: Yes
Specify libraries which should be ignored by Library Dependency Finder.

The correct value for this option is a library name (not folder name). You will see these names in “Library Dependency Graph” when building a project between < and > symbols.

**Example:**

```
Build output
...
LDF MODES: FINDER(chain+) COMPATIBILITY(soft)
Collected 54 compatible libraries
Scanning dependencies...
Dependency Graph
|-- <Hash> v1.0
  |-- <AsyncMqttClient> v0.8.2
  |-- <ESPAsyncTCP> v1.1.3
  |-- <ESP8266WiFi> v1.0
  |-- <ESP Async WebServer> v1.1.1
  |  |-- <ESPAsyncTCP> v1.1.3
  |  |-- <ESP8266WiFi> v1.0
  |  |-- <Hash> v1.0
  |  |-- <ArduinoJson> v5.13.1
  |  |-- <ESP8266WiFi> v5.13.1
  |  |-- <DNSServer> v1.1.0
  |  |-- <Ticker> v1.0
  |-- <Ticker> v1.0
....
```

`platformio.ini`

```
[env:myenv]
; Single line
lib_ignore = AsyncMqttClient, DNSServer

; Multi-line
lib_ignore =
  AsyncMqttClient
  ESP Async WebServer
```

**lib_extra_dirs**

See also:

Please make sure to read *Library Dependency Finder (LDF)* guide first.

Type: DirPath | Multiple: Yes

A list with extra directories/storages where *Library Dependency Finder (LDF)* will look for dependencies.

This option can also be set by global environment variable `PLATFORMIO_LIB_EXTRA_DIRS`.

**Warning:** This is a not direct path to a library with source code. It should be a path to storage that contains libraries grouped by folders. For example, `D:\PlatformIO\extra\libraries` but not `D:\PlatformIO\extra\libraries\FooLibrary`. 
Example:

```
[env:myenv]
lib_extra_dirs =
   /common/libraries
   /iot/libraries
```

**lib_ldf_mode**

See also:

Please make sure to read *Library Dependency Finder (LDF)* guide first.

Type: String | Multiple: No | Default: chain

This option specifies how does Library Dependency Finder should analyze dependencies (#include directives). See *Dependency Finder Mode* for details and available options.

Example:

```
[env:myenv]
; evaluate C/C++ Preprocessor conditional syntax
lib_ldf_mode = chain+
```

**lib_compat_mode**

See also:

Please make sure to read *Library Dependency Finder (LDF)* guide first.

Type: String | Multiple: No | Default: soft

Library compatibility mode allows one to control strictness of Library Dependency Finder. See *Compatibility Mode* for details and available options.

By default, this value is set to `lib_compat_mode = soft` and means that LDF will check only for framework compatibility.

Example:

```
[env:myenv]
; Checks for the compatibility with frameworks and dev/platforms
lib_compat_mode = strict
```

**lib_archive**

Type: Bool (yes or no) | Multiple: No | Default: yes

Create an archive (*.a, static library) from the object files and link it into a firmware (program). This is default behavior of PlatformIO Build System (`lib_archive = yes`).

Setting `lib_archive = no` will instruct PlatformIO Build System to link object files directly (in-line). This could be useful if you need to override weak symbols defined in framework or other libraries.

You can disable library archiving per a custom library using `libArchive` field in `library.json` manifest.

Example:
**Upload options**

- **upload_port**
- **upload_protocol**
- **upload_speed**
- **upload_flags**
- **upload_resetmethod**
- **upload_command**

**upload_port**

Type: String (Pattern) | Multiple: No

This option is used by "uploader" tool when sending firmware to board via `upload_port`. For example,

- `/dev/ttyUSB0` - Serial port (Unix-based OS)
- `COM3` - Serial port (Windows OS)
- `192.168.0.13` - IP address when using OTA
- `/media/disk` - physical path to media disk/flash drive (*Mbed* enabled boards)
- `D:` - physical path to media disk/flash drive (Windows OS).

If `upload_port` isn’t specified, then *PlatformIO* will try to detect it automatically.

To print all available serial ports please use `pio device list` command.

This option can also be set by global environment variable `PLATFORMIO_UPLOAD_PORT`.

Please note that you can use Unix shell-style wildcards:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches everything</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
<tr>
<td><code>[seq]</code></td>
<td>matches any character in seq</td>
</tr>
<tr>
<td><code>[^seq]</code></td>
<td>matches any character not in seq</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
[env:uno]
platform = atmelavr
framework = arduino

; any port that starts with /dev/ttyUSB
upload_port = /dev/ttyUSB*
```
upload_port

Type: String | Multiple: No

A protocol that “uploader” tool uses to talk to a board. Please check Boards for supported uploading protocols by your board.

Note: upload_protocol = custom allows one to use a custom upload_command - see below.

upload_speed

Type: Integer | Multiple: No

A connection speed (baud rate) which “uploader” tool uses when sending firmware to board.

upload_flags

Type: String | Multiple: Yes

Extra flags for uploader. Will be added to the end of uploader command. If you need to override uploader command or base flags please use extra_scripts.

This option can also be set by global environment variable PLATFORMIO_UPLOAD_FLAGS.

Example

Please specify each flag/option in a new line starting with minimum 2 spaces.

```ini
[env:atmega328pb]
platform = atmelavr
board = atmega328pb
framework = arduino
upload_flags =
    -P$UPLOAD_PORT
    -b$UPLOAD_SPEED
    -u
    -Ulock:w:0xCF:m
    -Uhfuse:w:0xD7:m
    -Uefuse:w:0xF6:m
    -Ulfuse:w:0xE2:m
```

upload_ResetMethod

Type: String | Multiple: No

Specify reset method for “uploader” tool. This option isn’t available for all development platforms. The only Espressif 8266 supports it.
**upload_command**

Type: String | Multiple: No

Override default `Development Platforms` upload command with a custom command. You can pass a full upload
command with arguments and options or mix with `upload_flags`.

In order to use `upload_command`, `upload_protocol = custom` must be specified.

Default upload commands are declared in `build/main.py` script file of `Development Platforms`. See a list with
open source `Development Platforms` => https://github.com/topics/platformio-platform

**Note:** Please note that you can use build variables in `upload_command`, such as PlatformIO project folders and
other runtime configuration. A list with build variables are available by running `pio run --target envdump` command.

**Examples**

1. Override default upload command but handle pre-uploading actions (looking for serial port, extra image prepara-
tion, etc.). Normally, the pre-configured upload options will be stored in `$UPLOADERFLAGS` build vari-
able. A classic default upload command for `Development Platforms` may look as `some-flash-bin-tool
$UPLOADERFLAGS $SOURCE`, where `$SOURCE` will be replaced by a real program/firmware binary.

```
$PROJECT_PACKAGES_DIR` build variable points to `packages_dir`
```

```
[env:program_via_AVR_ISP]
platform = atmelavr
framework = arduino
board = uno
upload_protocol = custom
upload_flags =
  -C
  $PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
  -p
  atmega328p
  -P $UPLOAD_PORT
  -b
  115200
  -c
  stk500v1
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i
```

2. Override default upload command and skip pre-uploading actions.

```
[env:program_via_usbasp]
platform = atmelavr
framework = arduino
board = uno
upload_protocol = custom
upload_flags =
  -C
  $PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
  -p
  atmega328p
  -Pusb
  -c
```
(continues on next page)
stk500v1
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i

; Use ST-util for flashing
; https://github.com/texane/stlink

[env:custom_st_flash]
platform = ststm32
framework = stm32cube
board = bluepill_f103c6
upload_protocol = custom
upload_command = →$
PROJECT_PACKAGES_DIR/tool-stlink/st-flash write $SOURCE 0x8000000

Monitor options

- monitor_port
- monitor_speed
- monitor_filters
- monitor_rts
- monitor_dtr
- monitor_flags

Custom options for pio device monitor command.

monitor_port

Type: String | Multiple: No

Port, a number or a device name, or valid URL Handlers. See pio device monitor --port. To print all available serial ports please use pio device list command.

Please note that you can use Unix shell-style wildcards:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches everything</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
<tr>
<td>[seq]</td>
<td>matches any character in seq</td>
</tr>
<tr>
<td>[!seq]</td>
<td>matches any character not in seq</td>
</tr>
</tbody>
</table>

Example:

[env:custom_monitor_port]
...
; Unix
monitor_port = /dev/ttyUSB1

(continues on next page)
monitor_port  =  COM[13]

; Socket
monitor_port  =  socket://localhost:4444

**monitor_speed**

Type: Integer | Multiple: No | Default: 9600

A monitor speed (baud rate). See `pio device monitor --baud`.

Example:

```
[env:custom_monitor_speedrate]
...
monitor_speed  =  115200
```

**monitor_filters**

Type: String | Multiple: Yes

Apply filters and text transformation for device output. See available filters at `Filters`.

Example:

```
[env:log_output_to_file_with_timestamp]
...
platform  =  ...
monitor_filters  =  log2file, time, default
```

**monitor_rts**

Type: Integer (0 or 1) | Multiple: No

A monitor initial RTS line state. See `pio device monitor --rts`.

**monitor_dtr**

Type: Integer (0 or 1) | Multiple: No

A monitor initial DTR line state. See `pio device monitor --dtr`.

**monitor_flags**

Type: String | Multiple: Yes

Pass extra flags and options to `pio device monitor` command. Please note that each flag, option or its value should be passed in a new line. See example below.

Available flags and options are the same which are documented for `pio device monitor` command.
Example:

```ini
[env:extra_monitor_flags]
platform = ...
board = ...
monitor_flags=
    --parity
    N
    --encoding
    hexlify
```

**Check options**

See also:

Please make sure to read *Static Code Analysis* guide first.

- `check_tool`
- `check_patterns`
- `check_flags`
- `check_severity`

**check_tool**

**Type:** String | **Multiple:** Yes | **Default:** cppcheck

A name of the check tool used for analysis. This option is useful when you want to check source code with two or more tools.

See available tools in *Check tools*.

**Example**

```ini
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck, clangtidy
```

**check_patterns**

**Type:** String (Pattern) | **Multiple:** Yes

This option allows specifying which source files or folders should be included/excluded from the check process. GLOB Patterns are allowed. `src_dir` and `include_dir` folders are checked by default.

Another option for filtering source files is `pio check --pattern` command.

**Example**

```ini
[env:myenv]
platform = ...
board = ...
check_patterns
```
[env:custom_check_patterns]
platform = ...
board = ...
check_tool = clangtidy
check_patterns =
   app/sources
   tests/hardware/*.c

check_flags

Type: String | Multiple: Yes

Additional flags to be passed to the tool command line. This option is useful when you want to adjust the check process to fit your project requirements. By default, the flags are passed to all tools specified in `check_tool` section. To set individual flags, define tool name at the beginning of the line.

Another option for adding flags is `pio check --flags` command.

Example

[env:extra_check_flags]
platform = ...
board = ...
check_tool = cppcheck, clangtidy
check_flags =
   --common-flag
   cppcheck: --enable=performance --inline-suppr
   clangtidy: -fix-errors -format-style=mozilla

check_severity

Type: String | Multiple: Yes | Default: low, medium, high

This option allows specifying the Defect severity types which will be reported by the Check tools.

Another option for filtering source files is `pio check --severity` command.

Example

[env:detect_only_medium_or_high_defects]
platform = ...
board = ...
check_severity = medium, high

Test options

See also:

Please make sure to read `Unit Testing` guide first.

- test_filter
- test_ignore
test_port

• test_port
• test_speed
• test_transport
• test_build_project_src

---

test_filter

Type: String (Pattern) | Multiple: Yes

Process only the *Unit Testing* tests where the name matches specified patterns.
Also, you can filter some tests using `pio test --filter` command.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches everything</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
<tr>
<td>[seq]</td>
<td>matches any character in seq</td>
</tr>
<tr>
<td>![seq]</td>
<td>matches any character not in seq</td>
</tr>
</tbody>
</table>

Example

```plaintext
[env:myenv]
test_filter = footest, bartest_*, test[13]
```

test_ignore

Type: String (Pattern) | Multiple: Yes

Ignore *Unit Testing* tests where the name matches specified patterns.
Also, you can ignore some tests using `pio test --ignore` command.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches everything</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
<tr>
<td>[seq]</td>
<td>matches any character in seq</td>
</tr>
<tr>
<td>![seq]</td>
<td>matches any character not in seq</td>
</tr>
</tbody>
</table>

Example

```plaintext
[env:myenv]
test_ignore = footest
             bartest_*
             test[13]
```

test_port

Type: String (Pattern) | Multiple: No
This option specifies communication interface (Serial/UART) between PlatformIO Unit Testing Engine and target device. For example,

- /dev/ttyUSB0 - Unix-based OS
- COM3 - Windows OS

If test_port isn’t specified, then PlatformIO will try to detect it automatically.

To print all available serial ports use pio device list command.

**test_speed**

Type: Integer | Multiple: No | Default: 115200

A connection speed (baud rate) to communicate with a target device.

**test_transport**

Type: String | Multiple: No

A transport type which will be used to read test results from a target device. See more details at Test Transport.

**test_build_project_src**

Type: Bool {yes or no} | Multiple: No | Default: no

Force Unit Testing engine to build project source code from src_dir setting test_build_project_src to yes. More detail about Shared Code.

**Example**

```
[env:myenv]
platform = ...

test_build_project_src = yes
```

**Debugging options**

See also:

Please make sure to read Debugging guide first.
debug_tool

Type: String | Multiple: No
A name of debugging tool. This option is useful when board supports more than one debugging tool (adapter, probe) or you want to create Custom debugging configuration.

See available tools in Tools & Debug Probes.

Example

```ini
[env:debug]
platform = ...
board = ...
debug_tool = custom
```

debug_build_flags

Type: String | Multiple: Yes | Default: -Og -g2 -ggdb2
These flags/options affect the preprocessing, compilation, assembly and linking processes for C and C++ code.

Note: This option might be helpful to adjust the optimization level if firmware with debug information is too big to be uploaded to a target

Example

```ini
[env:debug]
platform = ...
board = ...

; Set optimization level and amount of debug information generated by the compiler
debug_build_flags = -O0 -ggdb3 -g3
```

Other possible flags that might be useful for debugging your application.

debug_init_break

Type: String | Multiple: No | Default: tbreak main
An initial breakpoint that makes your program stop whenever a certain point in the program is reached. Default value is set to tbreak main and means creating a temporary breakpoint at int main(...) function and automatically delete it after the first time a program stops there.

• GDB Setting Breakpoints
• GDB Breakpoint Locations
Note: Please note that each debugging tool (adapter, probe) has limited number of hardware breakpoints. If you need more Project Initial Breakpoints, please place them in debug_extra_cmds.

Examples

```yaml
[env:debug]
platform = ...
board = ...

debug_init_break =

; Examples 1: disable initial breakpoint
debug_init_break =

; Examples 2: temporary stop at `void loop()`` function
debug_init_break = tbreak loop

; Examples 3: stop in main.cpp at line 13
debug_init_break = break main.cpp:13

; Examples 4: temporary stop at `void Reset_Handler(void)`
debug_init_break = tbreak Reset_Handler
```

debug_init_cmds

Type: String | Multiple: Yes | Default: See details...

Initial commands that will be passed to back-end debugger.

PlatformIO dynamically configures back-end debugger depending on a debug environment. Here is a list with default initial commands for the popular Tools & Debug Probes.

For example, the custom initial commands for GDB:

```yaml
[env:debug]
platform = ...
board = ...
debug_init_cmds =
    target extended-remote $DEBUG_PORT
    $INIT_BREAK
    monitor reset halt
    $LOAD_CMDS
    monitor init
    monitor reset halt
```

debug_extra_cmds

Type: String | Multiple: Yes

Extra commands that will be passed to back-end debugger after debug_init_cmds. For example, add custom breakpoint and load .gdbinit from a project directory for GDB:

```yaml
[env:debug]
platform = ...
board = ...
```
debug_extra_cmds =
    break main.cpp:13
    break foo.cpp:100
    source .gdbinit

Note: Initial Project Breakpoints: Use `break path/to/file:LINE_NUMBER` to define initial breakpoints for debug environment. Multiple breakpoints are allowed.

To save session breakpoints, please use `save breakpoints [filename]` command in Debug Console. For example, `save breakpoints .gdbinit`. Later, this file could be loaded via `source [filename]` command. See above.

debug_load_cmds

Type: String | Multiple: Yes | Default: load

Specify a command which will be used to load program/firmware to a target device. Possible options:

- **load - default** option
- **load [address]** - load program at specified address, where “[address]” should be a valid number
- **preload** - some embedded devices have locked Flash Memory (a few Freescale Kinetis and NXP LPC boards). In this case, firmware loading using debugging client is disabled. `preload` command instructs PlatformIO Core (CLI) to load program/firmware using development platform “upload” method (via bootloader, media disk, etc)
- (empty value, `debug_load_cmds =`), disables program loading at all.
- **custom commands** - pass any debugging client command (GDB, etc.)

Sometimes you need to run extra monitor commands (on debug server side) before program/firmware loading, such as flash unlocking or erasing. In this case we can combine service commands with loading and run them before. See example:

```ini
[env:debug]
platform = ...
board = ...
debug_load_cmds =
    monitor flash erase_sector 0 0 11
    load
```

debug_load_mode

Type: String | Multiple: No | Default: always

Allows one to control when PlatformIO should load debugging firmware to the end target. Possible options:

- **always** - load for the each debugging session, **default**
- **modified** - load only when firmware was modified
- **manual** - do not load firmware automatically. You are responsible to pre-flash target with debugging firmware in this case.

1.6. “platformio.ini” (Project Configuration File) 287
debug_server

Type: String | Multiple: Yes

Allows one to setup a custom debugging server. By default, boards are pre-configured with a debugging server that is compatible with “on-board” debugging tool (adapter, probe). Also, this option is useful for a Custom debugging tool.

Option format (multi-line):

- First line is an executable path of debugging server
- 2-nd and the next lines are arguments for executable file

Example:

```plaintext
[env:debug]
platform = ...
board = ...
debug_server = /path/to/debugging/server
arg1
arg2
...
argN
```

debug_port

Type: String | Multiple: No

A debugging port of a remote target. Could be a serial device or network address. PlatformIO detects it automatically if is not specified.

For example:

- /dev/ttyUSB0 - Unix-based OS
- COM3 - Windows OS
- localhost:3333

debug_svd_path

Type: FilePath | Multiple: No

A custom path to SVD file which contains information about device peripherals.

Advanced options

extends

Type: String | Multiple: Yes

This option allows one to inherit configuration from other sections or build environments in “platformio.ini” (Project Configuration File). Multiple items are allowed, split them with , or with a new line.

If you need to extend only a few options from some section, please take a look at Dynamic variables.

Example:
extra_scripts

Type: FilePath | Multiple: Yes

A list of PRE and POST extra scripts.

See details and examples in Advanced Scripting section.

If you plan to share these scripts with Remote Development machine, please put them to shared_dir.

1.6.3 Build Configurations

There are 2 types (build_type) of build configuration in PlatformIO:

release Default configuration. A “release” configuration of your firmware/program does not contain symbolic debug information and is optimized for the firmware size or speed (depending on Development Platforms)

debug A “debug” configuration of your firmware/program is compiled with full symbolic debug information and no optimization. Optimization complicates debugging, because the relationship between source code and generated instructions is more complex.

Note: If you need to control build flags that are specific for debug configuration please refer to debug_build_flags.

If you need to build a project in debug configuration, please use one of these options:

- Add build_type with debug value to “platformio.ini” (Project Configuration File)
- Use target debug for the pio run --target command.

Note: Debugging automatically switches to debug configuration when you do project debugging from PlatformIO IDE or use the pio debug command.

To avoid having Debugging rebuild the project, please create a separate build environment that defines build_type = debug. See the example below where the mydebug build environment will be used automatically by Debugging:
Please note that you can set a default build environment per a project using the *default_envs* option in Section [platformio].

### 1.6.4 Dynamic variables

Dynamic variables (interpolations) are useful when you have a custom configuration data between build environments. For examples, extra *build_flags* or project dependencies *lib_deps*.

Each variable should have a next format: `@{<section>.<option>}`, where `<section>` is a value from `[@<section>]` group, and `<option>` is a first item from pair `<option> = value`.

You can inject system environment variable using *sysenv* as a section. For example, `@{sysenv.HOME}`.

- Variable can be applied only for the option’s value
- Multiple variables are allowed
- The Section [platformio] and Section [env] sections are reserved and could not be used as a custom section. Some good section names might be *extra* or *custom*.

**Note:** If you need to share common configuration options between build environments, please take a look at “Global scope” in Section [env] or *extends* option which allows extending of other sections.

Example:

```plaintext
; You MUST inject these options into [env:] section
; using ${extra.***} (see below)
[extra]
build_flags = -D VERSION=1.2.3 -D DEBUG=1
lib_deps_builtin =
  SPI
  Wire
lib_deps_external =
  bblanchon/ArduinoJson@>5.6.0

[env:uno]
platform = atmelavr
framework = arduino
board = uno
build_flags = ${extra.build_flags}
lib_deps =
  ${extra.lib_deps_builtin}
```

(continues on next page)
```ini
[env:nodemcuv2]
platform = espressif8266
framework = arduino
board = nodemcuv2
build_flags = ${extra.build_flags} -Wall
lib_deps =
    ${extra.lib_deps_builtin}
    ${extra.lib_deps_external}
    knolleary/PubSubClient @ ~2.6
    paulstoffregen/OneWire @ ^2.3.5

; Keep sensitive data in environment variables
;
; Unix
; export WIFI_SSID='"my ssid name"'
; export WIFI_PASS='"my password"'
;
; Windows
; set WIFI_SSID='"my ssid name"
; set WIFI_PASS='"my password"

[env:esp32dev]
extends = env:nodemcuv2
platform = espressif32
board = esp32dev
build_flags =
    -DWIFI_SSID=${sysenv.WIFI_SSID}
    -DWIFI_PASS=${sysenv.WIFI_PASS}
```

**Warning:** Be careful with special characters in system environment variables on Unix systems, especially when they are used as the value for preprocessor directives. Symbols like $, &, ~, etc must be explicitly escaped, for example:

```ini
export WIFI_PASS='"my\-p\&a\\$\\\$\$\\\$word"'
```

### 1.6.5 Examples

**Note:** A full list with project examples can be found in PlatformIO Repository.

Community project examples with `platformio.ini`:

- MarlinFirmware/Marlin
- xoseperez/espurna
- esphome/esphome
- cyberman54/ESP32-Paxcounter
- 1technophile/OpenMQTTGateway

**Example**
For more examples, see *Examples*.

```python
[platformio]
default_envs = nodemcu

; You MUST inject these options into [env:] section
; using $common_env_data.*** (see below)
[common_env_data]
build_flags =
  -D VERSION=1.2.3
  -D DEBUG=1
lib_deps_builtin =
  SPI
  Wire
lib_deps_external =
  ArduinoJson@~5.6,!=5.4
  https://github.com/gioblu/PJON.git#v2.0
  IRremoteESP8266=https://github.com/markszabo/IRremoteESP8266/archive/master.zip

[env:nodemcu]
platform = espressif8266
framework = arduino
board = nodemcu

; Build options
build_flags =
  ${common_env_data.build_flags}
  -DSSID_NAME=HELLO
  -DSSID_PASSWORD=WORLD

; Library options
lib_deps =
  ${common_env_data.lib_deps_builtin}
  ${common_env_data.lib_deps_external}
  https://github.com/me-no-dev/ESPAsyncTCP.git
  PubSubClient@2.6
  OneWire

; Serial Monitor options
monitor_speed = 115200
monitor_flags =
  --encoding
  hexlify

; Unit Testing options
test_ignore = test_desktop

[env:bluepill_f103c8]
platform = ststm32
framework = arduino
board = bluepill_f103c8

; Build options
build_flags = ${common_env_data.build_flags}

; Library options
lib_deps =
  ${common_env_data.lib_deps_external}
```

(continues on next page)
; Debug options
debug_tool = custom
ddebug_server =
   JLinkGDBServer
   -singlerun
   -if
   SWD
   -select
   USB
   -port
   2331
   -device
   STM32F103C8

; Unit Testing options
test_ignore = test_desktop

1.7 Environment variables

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. PlatformIO handles variables which start with PLATFORMIO_ prefix.

How to set environment variable?

# Windows
set VARIABLE_NAME=VALUE

# Windows GUI -> https://www.youtube.com/watch?v=bEroNNzqlF4

# Unix (bash, zsh)
export VARIABLE_NAME=VALUE

# Unix (fish)
set -x VARIABLE_NAME VALUE

1.7.1 General

PlatformIO uses General environment variables for the common operations/commands.
PlatformIO handles CI variable which is setup by Continuous Integration (Travis, Circle and etc.) systems. PlatformIO uses it to disable prompts and progress bars. In other words, CI=true automatically setup `PLATFORMIO_DISABLE_PROGRESSBAR` to true.

**PLATFORMIO Auth Token**

Allows one to specify Personal Authentication Token that could be used for automatic login into PlatformIO Account. It is very useful for Continuous Integration systems and Remote Development operations where you are not able manually authorize.

You can get own Personal Authentication Token using `pio account token` command.

**PLATFORMIO_FORCE_ANSI**

Force to output ANSI control character even if the output is a pipe (not a tty). The possible values are true and false. Default is `PLATFORMIO_FORCE_ANSI=false`.

**PLATFORMIO_NO_ANSI**

Do not print ANSI control characters. The possible values are true and false. Default is `PLATFORMIO_NO_ANSI=false`.

You can also use `pio --no-ansi` flag for PlatformIO Core (CLI).

**PLATFORMIO_DISABLE_PROGRESSBAR**

Disable progress bar for package/library downloader and uploader. This is useful when calling PlatformIO from subprocess and output is a pipe (not a tty). The possible values are true and false. Default is `PLATFORMIO_DISABLE_PROGRESSBAR=false`.

### 1.7.2 Directories

**PLATFORMIO_CORE_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `core_dir`.

It may need to re-install PlatformIO Core (CLI) (remove default core directory) to take effect.

**PLATFORMIO_GLOBALLIB_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `globallib_dir`.

**PLATFORMIO_PLATFORMS_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `platforms_dir`.

**PLATFORMIO_PACKAGES_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `packages_dir`.

**PLATFORMIO_CACHE_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `cache_dir`.

**PLATFORMIO_BUILD_CACHE_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `build_cache_dir`.

**PLATFORMIO_WORKSPACE_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `workspace_dir`.

**PLATFORMIO_INCLUDE_DIR**

Allows one to override "platformio.ini" (Project Configuration File) option `include_dir`.

**PLATFORMIO_SRC_DIR**

---

Chapter 1. Contents
Allows one to override “platformio.ini” (Project Configuration File) option src_dir.

**PLATFORMIO_LIB_DIR**

Allows one to override “platformio.ini” (Project Configuration File) option lib_dir.

**PLATFORMIO_LIBDEPS_DIR**

Allows one to override “platformio.ini” (Project Configuration File) option libdeps_dir.

**PLATFORMIO_BUILD_DIR**

Allows one to override “platformio.ini” (Project Configuration File) option build_dir.

**PLATFORMIO_DATA_DIR**

Allows one to override “platformio.ini” (Project Configuration File) option data_dir.

**PLATFORMIO_TEST_DIR**

Allows one to override “platformio.ini” (Project Configuration File) option test_dir.

**PLATFORMIO_BOARDS_DIR**

Allows one to override “platformio.ini” (Project Configuration File) option boards_dir.

**PLATFORMIO_SHARED_DIR**

Allows one to override “platformio.ini” (Project Configuration File) option shared_dir.

**PLATFORMIO_REMOTE_AGENT_DIR**

Allows one to override pio remote agent start --working-dir.

**PLATFORMIO_LIB_EXTRA_DIRS**

Allows one to set “platformio.ini” (Project Configuration File) option lib_extra_dirs.

### 1.7.3 Building

**PLATFORMIO_BUILD_FLAGS**

Allows one to set “platformio.ini” (Project Configuration File) option build_flags.

Examples:

```bash
# Unix:
export PLATFORMIO_BUILD_FLAGS=-DFOO
export PLATFORMIO_BUILD_FLAGS=-DFOO -DBAR=1 -Wall

# Windows:
SET PLATFORMIO_BUILD_FLAGS=-DFOO
SET PLATFORMIO_BUILD_FLAGS=-DFOO -DBAR=1 -Wall
```

**Warning:** Consider using *Dynamic variables* instead of PLATFORMIO_BUILD_FLAGS environment variable if additional build flags contain preprocessor directive with special characters ($, &, ~, etc) in its value.

**PLATFORMIO_SRC_BUILD_FLAGS**

Allows one to set “platformio.ini” (Project Configuration File) option src_build_flags.

**PLATFORMIO_SRC_FILTER**

### 1.7. Environment variables
Allows one to set “platformio.ini” (Project Configuration File) option `src_filter`.

**PLATFORMIO_EXTRA_SCRIPTS**

Allows one to set “platformio.ini” (Project Configuration File) option `extra_scripts`.

**PLATFORMIO_DEFAULT_ENVS**

Allows one to set “platformio.ini” (Project Configuration File) option `default_envs`.

### 1.7.4 Uploading

**PLATFORMIO_UPLOAD_PORT**

Allows one to set “platformio.ini” (Project Configuration File) option `upload_port`.

**PLATFORMIO_UPLOAD_FLAGS**

Allows one to set “platformio.ini” (Project Configuration File) option `upload_flags`.

### 1.7.5 Settings

Allows one to override PlatformIO settings. You can manage them via `pio settings` command.

**PLATFORMIO_SETTING_AUTO_UPDATE_LIBRARIES**

Allows one to override setting `auto_update_libraries`.

**PLATFORMIO_SETTING_AUTO_UPDATE_PLATFORMS**

Allows one to override setting `auto_update_platforms`.

**PLATFORMIO_SETTING_CHECK_LIBRARIES_INTERVAL**

Allows one to override setting `check_libraries_interval`.

**PLATFORMIO_SETTING_CHECK_PLATFORMIO_INTERVAL**

Allows one to override setting `check_platformio_interval`.

**PLATFORMIO_SETTING_CHECK_PLATFORMS_INTERVAL**

Allows one to override setting `check_platforms_interval`.

**PLATFORMIO_SETTING_ENABLE_CACHE**

Allows one to override setting `enable_cache`.

**PLATFORMIO_SETTING_ENABLE_TELEMETRY**

Allows one to override setting `enable_telemetry`.

**PLATFORMIO_SETTING_FORCE_VERBOSE**

Allows one to override setting `force_verbose`.

**PLATFORMIO_SETTING_PROJECTS_DIR**

Allows one to override setting `projects_dir`. 
1.8 Advanced Scripting

**Warning:** Advanced Scripting is recommended for Advanced Users and requires knowledge of the Python language.

**Warning:** *Dynamic build flags* is a highly recommended alternative to advanced scripting, where you can use any programming language. Also, that option is useful if you need to apply changes to the project before the building/uploading process, such as:

- Macro with the latest VCS revision/tag “on-the-fly”
- Generate dynamic headers (*.h)*
- Process media content before generating SPIFFS image
- Make some changes to source code or related libraries

**Contents**

- Advanced Scripting
  - Launch types
  - Construction Environments
  - Before/Pre and After/Post actions
  - Build Middlewares
  - Custom Targets
    - Build System API
    - Examples
      - Command shortcut
      - Dependent target
      - Target with options
  - Other Use Cases
    - Custom options in `platformio.ini`
    - Split C/C++ build flags
    - Extra Linker Flags without `-Wl`, prefix
    - Custom upload tool
    - Upload to Cloud (OTA)
    - Custom firmware/program name
    - Override package files
    - Override Board Configuration
    - Custom debug flags
* Extra Python packages

The PlatformIO Build System allows the user to extend the build process with custom scripts using the Python interpreter and the SCons construction tool. Build flags, upload flags, targets, toolchains data and other information are available for modification as SCons Construction Environments. Custom scripts are included with `extra_scripts`

**Warning:** You cannot run or debug these scripts manually with a Python interpreter. They will be loaded automatically when the `pio run` command processes the project environment.

### 1.8.1 Launch types

There are two execution orders for extra scripts:

1. **PRE** - executes before the main script of *Development Platforms*
2. **POST** - executes after the main script of *Development Platforms*

Multiple extra scripts are allowed. Please split them via “,” (comma + space) in the same line or use multi-line values.

For example, in “`platformio.ini`” (Project Configuration File):

```ini
[env:my_env_1]
platform = ...
; Defaults to POST script since no prefix is used
extra_scripts = post_extra_script.py

[env:my_env_2]
platform = ...
extra_scripts =
    pre:pre_extra_script.py
    post:post_extra_script1.py
    post:post_extra_script2.py
```

This option can also be set by the global environment variable `PLATFORMIO_EXTRA_SCRIPTS`.

### 1.8.2 Construction Environments

The PlatformIO Build System uses two built-in construction environments to process each project:

- `env, Import("env")` - the global construction environment used for the *Development Platforms* and Framesworks build scripts, upload tools, *Library Dependency Finder (LDF)*, and other internal operations
- `projenv, Import("projenv")` - the isolated construction environment used for processing the project source code in `src_dir`. Please note that any `src_build_flags` specified in “`platformio.ini`” (Project Configuration File) will be passed to `projenv` and not to `env`.

**Warning:**

1. `projenv` is available only for POST-type scripts
2. Flags passed to `env` using PRE-type script will affect `projenv` too.

`my_pre_extra_script.py:`
Import("env")

# access to global construction environment
print(env)

# Dump construction environment (for debug purpose)
print(env.Dump())

# append extra flags to global build environment
# which later will be used to build:
#  - project source code
#  - frameworks
#  - dependent libraries
env.Append(CPPDEFINES=[
   "MACRO_1_NAME",
   ("MACRO_2_NAME", "MACRO_2_VALUE")
])

my_post_extra_script.py:

Import("env", "projenv")

# access to global construction environment
print(env)

# access to project construction environment
print(projenv)

# Dump construction environments (for debug purpose)
print(env.Dump())
print(projenv.Dump())

# append extra flags to global build environment
# which later will be used to build:
#  - frameworks
#  - dependent libraries
env.Append(CPPDEFINES=[
   "MACRO_1_NAME",
   ("MACRO_2_NAME", "MACRO_2_VALUE")
])

# append extra flags to only project build environment
projenv.Append(CPPDEFINES=[
   "PROJECT_EXTRA_MACRO_1_NAME",
   ("PROJECT_EXTRA_MACRO_2_NAME", "PROJECT_EXTRA_MACRO_2_VALUE")
])

See examples below how to import construction environments and modify existing data or add new.

1.8.3 Before/Pre and After/Post actions

The PlatformIO Build System has a rich API that allows one to attach different pre-/post actions (hooks) using env.AddPreAction(target, callback) or env.AddPreAction(target, [callback1, callback2, ...]) function. The first argument target can be the name of a target that is passed using the pio run --target command, the name of a built-in target (buildprog, size, upload, program, buildfs, uploadfs, uploadfsota) or the path to a file which PlatformIO processes (ELF, HEX, BIN, OBJ, etc.).
Examples

The `extra_script.py` file is located in the same directory as `platformio.ini`.

```ini
[env:pre_and_post_hooks]
extra_scripts = post:extra_script.py
```

```python
extra_script.py:
import "env", "projenv"

# access to global build environment
print(env)

# access to project build environment (is used source files in "src" folder)
print(projenv)

# Dump build environment (for debug purpose)
# print(env.Dump())

# Change build flags in runtime
env.ProcessUnFlags("-DVECT_TAB_ADDR")
env.Append(CPPDEFINES=("VECT_TAB_ADDR", 0x123456789))

# Upload actions

def before_upload(source, target, env):
    print("before_upload")
    # do some actions
    # call Node.JS or other script
    env.Execute("node --version")

def after_upload(source, target, env):
    print("after_upload")
    # do some actions

print("Current build targets", map(str, BUILD_TARGETS))

env.AddPreAction("upload", before_upload)
env.AddPostAction("upload", after_upload)

# Custom actions when building program/firmware
env.AddPreAction("buildprog", callback...)
env.AddPostAction("buildprog", callback...)
```

(continues on next page)
# Custom actions for specific files/objects

```python
env.AddPreAction("$BUILD_DIR/${PROGNAME}.elf", [callback1, callback2,...])
env.AddPostAction("$BUILD_DIR/${PROGNAME}.hex", callback...)
```

# custom action before building SPIFFS image. For example, compress HTML, etc.
```python
env.AddPreAction("$BUILD_DIR/spiffs.bin", callback...)
```

# custom action for project's main.cpp
```python
env.AddPostAction("$BUILD_DIR/src/main.cpp.o", callback...)
```

# Custom HEX from ELF
```python
env.AddPostAction(
    "$BUILD_DIR/${PROGNAME}.elf",
    env.VerboseAction("".join(
        "$OBJCOPY", "-0", "ihex", "-R", ".eeprom",
        "$BUILD_DIR/${PROGNAME}.elf", "$BUILD_DIR/${PROGNAME}.hex"
    ), "Building $BUILD_DIR/${PROGNAME}.hex"
)
)
```

## 1.8.4 Build Middlewares

PlatformIO Build System allows you to add middleware functions that can be used for Build Node(Object) construction. This is very useful if you need to add custom flags for the specific file nodes or exclude them from a build process.

There is `env.AddBuildMiddleware(callback, pattern)` helper which instructs PlatformIO Build System to call `callback` for each SCons File System Node whose path matches with Unix shell-style “pattern” (wildcards).

If a pattern is omitted, the callback will be called for each File System Node which is added for the build process.

You can add an unlimited number of build middlewares. They will be called in order of registration. Please note, if the first middleware ignores some File Nodes, they will not be passed to the next middleware in chain.

### Examples

**platformio.ini:**

```
[env:build_middleware]
extra_scripts = pre:extra_script.py
```

**extra_script.py:**

```python
import("env")

# --- Add custom macros for the ALL files which name contains "http"
def extra_http_configuration(node):
    ""
    "node.name" - a name of File System Node
    "node.get_path()" - a relative path
    "node.get_abspath()" - an absolute path
    ""
```

1.8. Advanced Scripting
# do not modify node if file name does not contain "http"
if "http" not in node.name:
    return node

# now, we can override ANY SCons variables (CPPDEFINES, CCFLAGS, etc.,) for the
# specific file
# pass SCons variables as extra keyword arguments to `env.Object()` function
# p.s: run `pio run -t envdump` to see a list with SCons variables
return env.Object(node,
    CPPDEFINES=env['CPPDEFINES']
    + [('HTTP_HOST', device.local), ('HTTP_PORT', 8080)],
    CCFLAGS=env['CCFLAGS'] + ['-fno-builtin-printf'])

env.AddBuildMiddleware(extra_http_configuration)

# --- Replace some file from a build process with another

def replace_node_with_another(node):
    return env.File(path/to/patched/RtosTimer.cpp)

env.AddBuildMiddleware(replace_node_with_another,
    "framework-mbed/rtos/RtosTimer.cpp")

# --- Skip assembly *.S files from build process

def skip_asm_from_build(node):
    # to ignore file from a build process, just return None
    return None

env.AddBuildMiddleware(skip_asm_from_build, "*.S")

### 1.8.5 Custom Targets

New in version 5.0.

PlatformIO allows you to declare unlimited number of the custom targets. There are a lot of use cases for them:

- Pre/Post processing based on a dependent sources (other target, source file, etc.)
- Command launcher with own arguments
- Launch command with custom options declared in “platformio.ini” (Project Configuration File)
- Python callback as a target (use the power of Python interpreter and PlatformIO Build API).

A custom target can be processed using `pio run --target` option and you can list them via `pio run --list-targets` command.
Build System API

Import("env")

env.AddCustomTarget(
    name,
    dependencies,
    actions,
    title=None,
    description=None,
    always_build=True
)

AddCustomTarget arguments:

- **name** A name of target. ASCII chars (a-z, 0-9, _, -) are recommended. Good names are “gen_headers”, “program_bitstream”, etc.
- **dependencies** A list of dependencies that should be built BEFORE target will be launched. It is possible to pass multiple dependencies as a Python list ["dep1", dep_target_2]. If a target does not have dependencies, None should be passed.
- **actions** A list of actions to call on a target. It is possible to pass multiple actions as a Python list ["python --version", my_callback].
- **title** A title of a target. It will be printed when using PlatformIO Core (CLI) or PlatformIO IDE. We recommend to keep a title very short, 1-2 words.
- **description** The same as a title argument but allows you to provide detailed explanation what target does.
- **always_build** If there are declared dependencies and they are already built, this target will not be called if always_build=False. A default value is always_build=True and means always building/calling target.

Examples

Command shortcut

Create a custom node target (alias) which will print a NodeJS version

```
platformio.ini:

[env:myenv]
platform = ...
...
extra_scripts = extra_script.py
```

extra_script.py:

```
import("env")

# Single action/command per 1 target
env.AddCustomTarget("sysenv", None, "python -c "import os; print(os.environ)""

# Multiple actions
env.AddCustomTarget(
```

(continues on next page)
Now, run `pio run --target sysenv` or `pio run -t pioenv` (short version).

**Dependent target**

Sometimes you need to run a command which depends on another target (file, firmware, etc). Let’s create an `ota` target and declare command which will depend on a project firmware. If a build process successes, declared command will be run.

**platformio.ini:**

```ini
[env:myenv]
platform = ...
...
extra_scripts = extra_script.py
```

**extra_script.py:**

```python
Import("env")

env.AddCustomTarget(
    "ota",
    "$BUILD_DIR/$SOURCE.elf",
    "ota_script --firmware-path $SOURCE"
)
```

Now, run `pio run -t ota`.

**Target with options**

Let’s create a simple `ping` target and process it with `pio run --target ping` command:

**platformio.ini:**

```ini
[env:env_custom_target]
platform = ...
...
extra_scripts = extra_script.py
custom_ping_host = google.com
```

**extra_script.py:**

```python
Import("env")

host = env.GetProjectOption("custom_ping_host")
```

(continues on next page)
**1.8.6 Other Use Cases**

The best examples are PlatformIO development platforms. Please check builder folder for the main and framework scripts.

**Custom options in platformio.ini**

PlatformIO allows you extending project configuration with own data. You can read these values later using ProjectConfig API:

- `ProjectConfig::get(section, option, default=None)`: Get an option value for the named section
- `ProjectConfig::options(section)`: Returns a list of the sections available
- `ProjectConfig::items(section, as_dict=False)`: Returns a list of “name”, “value” pairs for the options in the given section or a dictionary when as_dict=True is passed
- `ProjectConfig::has_section(section)`: Indicates whether the named section is present in the configuration
- `ProjectConfig::has_option(section, option)`: If the given section exists, and contains the given option, returns True; otherwise returns False.

PlatformIO’s “ProjectConfig” is compatible with a native Python’s ConfigParser API.

**Example**

platformio.ini:

```ini
[universe]
hello = world

[env:my_env]
platform = ...
extra_scripts = extra_script.py

custom_option1 = value1
custom_option2 = value2
```

eextra_script.py:

```python
# "env.GetProjectOption" shortcut for the active environment
value1 = env.GetProjectOption("custom_option1")
value2 = env.GetProjectOption("custom_option2")

# Read value from other environments
config = env.GetProjectConfig()
world = config.get("universe", "hello")
```
Split C/C++ build flags

platformio.ini:

```ini
[env:my_env]
platform = ...
extra_scripts = extra_script.py
```

eextra_script.py (place it near platformio.ini):

```python
import("env")

# General options that are passed to the C and C++ compilers
env.Append(CCFLAGS=['flag1', 'flag2'])

# General options that are passed to the C compiler (C only; not C++).
env.Append(CFLAGS=['flag1', 'flag2'])

# General options that are passed to the C++ compiler
env.Append(CXXFLAGS=['flag1', 'flag2'])
```

Extra Linker Flags without –Wl, prefix

Sometimes you need to pass extra flags to GCC linker without –Wl,. You could use build_flags option but it will not work. PlatformIO will not parse these flags to LINKFLAGS scope. In this case, simple extra script will help:

platformio.ini:

```ini
[env:env_extra_link_flags]
platform = windows_x86
extra_scripts = extra_script.py
```

eextra_script.py (place it near platformio.ini):

```python
import("env")

# Dump build environment (for debug)
# print(env.Dump())

env.Append(LINKFLAGS=['-static',
                      '-static-libgcc',
                      '-static-libstdc++'])
```

Custom upload tool

You can override default upload command of development platform using extra script. There is the common environment variable UPLOADCMD which PlatformIO Build System will handle when you `pio run -t upload`.

Please note that some development platforms can have more than 1 upload command. For example, Atmel AVR has UPLOADHEXCMD (firmware) and UPLOADEEPCMD (EEPROM data).
See examples below:

**Template**

```
platformio.ini:
```

```
[env:my_custom_upload_tool]
platform = ...
; place it into the root of project or use full path
extrascripts = extra_script.py
upload_protocol = custom
; each flag in a new line
uploadflags =
-arg1
-arg2
-argN
```


```
extra_script.py (place it near platformio.ini):
```

```
import("env")

# please keep $SOURCE variable, it will be replaced with a path to firmware

# Generic
env.Replace(
    UPLOADER="executable or path to executable",
    UPLOADCMD="$UPLOADER $UPLOADERFLAGS $SOURCE"
)

# In-line command with arguments
env.Replace(
    UPLOADCMD="executable -arg1 -arg2 $SOURCE"
)

# Python callback
def on_upload(source, target, env):
    print(source, target)
    firmware_path = str(source[0])
    # do something
    env.Execute("executable arg1 arg2")

env.Replace(UPLOADCMD=on_upload)
```

**Custom openOCD command**

```
platformio.ini:
```

```
[env:disco_f407vg]
platform = ststm32
board = disco_f407vg
framework = mbed
extra_scripts = extra_script.py
upload_protocol = custom
; each flag in a new line
uploadflags =
- $f
    scripts/interface/stlink.cfg
- $f
```

(continues on next page)
extra_script.py (place it near platformio.ini):

```python
import("env")

platform = env.PioPlatform()

env.Prepend(
    UPLOADERFLAGS=["-s", platform.get_package_dir("tool-openocd") or "]
)
env.Append(
    UPLOADERFLAGS=["-c", "program {{SOURCE}} verify reset; shutdown"]
)
env.Replace(
    UPLOADER=openocd,
    UPLOADCMD="$UPLOADER $UPLOADERFLAGS"
)
```

Upload to Cloud (OTA)

See project example https://github.com/platformio/bintray-secure-ota

Custom firmware/program name

Sometimes is useful to have a different firmware/program name in `build_dir`

platformio.ini:

```ini
[env:env_custom_prog_name]
platform = espressif8266
board = nodemcuv2
framework = arduino
build_flags = -D VERSION=13
extra_scripts = pre:extra_script.py
```

```python
import("env")

my_flags = env.ParseFlags(env['BUILD_FLAGS'])
defines = {k: v for (k, v) in my_flags.get("CPPDEFINES"))
# print(defines)
env.Replace(PROGNAME="firmware_%s" % defines.get("VERSION"))
```

Override package files

PlatformIO Package Manager automatically installs pre-built packages (Frameworks, toolchains, libraries) required by development Development Platforms and build process. Sometimes you need to override original files with own versions: configure custom GPIO, do changes to built-in LD scripts, or some patching to installed library dependency.

The simplest way is using Diff and Patch technique. How does it work?
1. Modify original source files
2. Generate patches
3. Apply patches via PlatformIO extra script before build process.

Example

We need to patch the original `standard/pins_arduino.h` variant from Arduino framework and add extra macro 

```diff
# define PIN_A8 (99)
```

Let's duplicate `standard/pins_arduino.h` and apply changes. Generate a patch file and place it into `patches` folder located in the root of a project:

```diff
diff ~/.platformio/packages/framework-arduinoavr/variants/standard/pins_arduino.h /→tmp/pins_arduino_modified.h > /path/to/platformio/project/patches/1-framework-→arduinoavr-add-pin-a8.patch
```

The result of `1-framework-arduinoavr-add-pin-a8.patch`:

```diff
63a64 > # define PIN_A8 (99)
112c113 < // 14-21 PA0-PA7 works
---
> // 14-21 PA0-PA7 works
```

Using extra scripting we can apply patching before a build process. The final result of “`platformio.ini`” (Project Configuration File) and “PRE” extra script named `apply_patches.py`:

```
platformio.ini:
```

```
[env:uno]
platform = atmelavr
board = uno
framework = arduino
extra_scripts = pre:apply_patches.py
```

```
apply_patches.py:
```

```
from os.path import join, isfile 

env.Execute("touch " + patchflag_path)
```

(continues on next page)
env.Execute(lambda *args, **kwargs: _touch(patchflag_path))

Please note that this example will work on a system where a patch tool is available. For Windows OS, you can use patch and diff tools provided by Git client utility (located inside installation directory).

If you need to make it more independent to the operating system, please replace the patch with a multi-platform python-patch script.

**Override Board Configuration**

PlatformIO allows one to override some basic options (integer or string values) using More options in “platformio.ini” (Project Configuration File). Sometimes you need to do complex changes to default board manifest and extra PRE scripting work well here. See example below how to override default hardware VID/PIDs.

**Warning:** Due to a technical limitation these board changes will not work for pio device monitor command.

platformio.ini:

```
[env:uno]
platform = atmelavr
board = uno
framework = arduino
extra_scripts = pre:custon_hwids.py
```

custon_hwids.py:

```
Import("env")

board_config = env.BoardConfig()
# should be array of VID:PID pairs
board_config.update("build.hwids", [
    ["0x2341", "0x0243"],  # 1st pair
    ["0x2A03", "0x0043"].  # 2nd pair, etc.
])
```

**Custom debug flags**

PlatformIO removes all debug/optimization flags before a debug session or when Build Configurations is set to debug and overrides them with -0g -g2 -ggdb2 for ASFLAGS,CCFLAGS, and LINKFLAGS build scopes.

An extra script allows us to override PlatformIO’s default behavior and declare custom flags. See example below where we override -Og with -00:

platformio.ini:

```
[env:teensy31]
platform = teensy
board = teensy31
framework = arduino
extra_scripts = custom_debug_flags.py
```

custom_debug_flags.py:
Extra Python packages

If your project depends on the extra Python packages, you can use extra script to install them into the same virtual environment where PlatformIO Core (CLI) is installed.

platformio.ini:

```
[env:my_env]
platform = ...
extra_scripts = extra_script.py
```

eextra_script.py (place it near platformio.ini):

```
import("env")

# List installed packages
env.Execute("$PYTHONEXE -m pip list")

# Install custom packages from the PyPi registry
env.Execute("$PYTHONEXE -m pip install pkg1 pkg2")
```

1.9 Library Management

PlatformIO Library Manager is a tool for managing libraries of PlatformIO Registry and VCS repositories (Git, Hg, SVN). It makes it exceedingly simple to find, install and keep libraries up-to-date. PlatformIO Library Manager supports Semantic Versioning and its rules.

PlatformIO IDE has built-in PlatformIO Home with a modern GUI which allows:

- Search for new libraries in PlatformIO Registry
- “1-click” library installation, per-project libraries, extra storages
- List installed libraries in multiple storages
- List built-in libraries (by frameworks)
- Updates for installed libraries
- Multiple examples, trending libraries, and more.
1.9.1 Quick Start

PlatformIO Library Manager is a tool for managing libraries of PlatformIO Registry and VCS repositories (Git, Hg, SVN). It makes it exceedingly simple to find, install and keep libraries up-to-date. PlatformIO Library Manager supports Semantic Versioning and its rules.

There are 3 options how to find/manage libraries:

- PlatformIO Home
- Web Library Search
- PlatformIO Core Command Line Interface

You can manage different library storages using `pio lib --global` or `pio lib --storage-dir` options. If you change current working directory in terminal to project folder, then `pio lib` command will manage automatically dependency storage in `libdeps_dir`.

Project dependencies

PlatformIO Library Manager allows one to specify project dependencies (`lib_deps`) that will be installed automatically per project before environment processing. You do not need to install libraries manually. The only one simple step is to define dependencies in “platformio.ini” (Project Configuration File). For example,

```
[env:myenv]
platform = ...
framework = ...
board = ...
lib_deps =
    knolleary/PubSubClient
    bblanchon/ArduinoJson @ ~5,!=5.4
    https://github.com/gioblu/PJON.git#v2.0
    https://github.com/me-no-dev/ESPAsyncTCP.git
    https://github.com/adafruit/DHT-sensor-library/archive/master.zip
```

Please follow to `pio lib install` for detailed documentation about possible values.

**Warning:** If some libraries are not visible in PlatformIO IDE and Code Completion or Code Linting does not work properly, please perform

- **VSCode:** “Menu: View > Command Palette… > PlatformIO: Rebuild C/C++ Project Index”

PlatformIO IDE

PlatformIO IDE has built-in PlatformIO Home with a modern GUI which allows:
PlatformIO Core

CLI Guide

1.9.2 Library Dependency Finder (LDF)

Library Dependency Finder is a core part of PlatformIO Build System that operates with the C/C++ source files and looks for `#include ...` directives to know what header directories to include for the compiler.

In spite of the fact that Library Dependency Finder is written in pure Python, it evaluates C/C++ Preprocessor conditional syntax (`#ifdef`, `if`, `defined`, `else`, and `elif`) without calling `gcc -E`. This approach allows to significantly reduce the total compilation time. See Dependency Finder Mode for more details.

Contents

- Library Dependency Finder (LDF)
  - Configuration
  - Storage
  - Dependency Finder Mode
  - Compatibility Mode
  - C/C++ Preprocessor conditional syntax

Configuration

Library Dependency Finder can be configured from “platformio.ini” (Project Configuration File):
Storage

There are different storages where Library Dependency Finder looks for libraries. These storages (folders) have priority and LDF operates in the next order:

1. `lib_extra_dirs` - extra storages per build environment
2. `lib_dir` - own/private library storage per project
3. `libdeps_dir` - project dependency storage used by Library Management
4. “core_dir/lib” - global storage per all projects.
5. Library storages built into frameworks, SDKs.

Dependency Finder Mode

Library Dependency Finder starts work from analyzing source files of the project (`src_dir`) and can work in the next modes:

- **off** “Manual mode”, does not process source files of a project and dependencies. Builds only the libraries that are specified in manifests (`library.json`, `module.json`) or using `lib_deps` option.
- **chain** [DEFAULT] Parses ALL C/C++ source files of the project and follows only by nested includes (`#include ...`, `chain...`) from the libraries. It also parses C, CC, CPP files from libraries which have the same name as included header file. **Does not evaluate C/C++ Preprocessor conditional syntax.**
- **deep** Parses ALL C/C++ source files of the project and parses ALL C/C++ source files of the each found dependency (recursively). **Does not evaluate C/C++ Preprocessor conditional syntax.**
- **chain+** The same behavior as for the chain but **evaluates C/C++ Preprocessor conditional syntax.**
- **deep+** The same behavior as for the deep but **evaluates C/C++ Preprocessor conditional syntax.**

The mode can be changed using `lib_ldf_mode` option in “platformio.ini” (Project Configuration File). Default value is set to chain.

Note: Usually, when the LDF appears to fail to identify a dependency of a library, it is because the dependency is only referenced from a library source file, and not a library header file (see example below). In this case, it is necessary to either explicitly reference the dependency from the project source or “platformio.ini” (Project Configuration File) (`lib_deps` option), or change the LDF mode to “deep” (not generally recommended).

A difference between chain/chain+ and deep/deep+ modes. For example, there are 2 libraries:

- Library Foo with files:
  - Foo/foo.h
  - Foo/foo.cpp
  - Foo/extra.cpp
- Library Bar with files:
  - Bar/bar.h
  - Bar/bar.cpp

**Case 1**

- `lib_ldf_mode = chain`
• Foo/foo.h depends on the Bar library (contains #include <bar.h>)
• #include <foo.h> is located in one of the project source files

Here the nested includes (project file > foo.h > bar.h) and LDF will find both libraries Foo and Bar.

Case 2

• lib_ldf_mode = chain
• Foo/extra.cpp depends on the Bar library (contains #include <bar.h>)
• #include <foo.h> is located in one of the project source files

In this case, LDF will not find the Bar library because it doesn’t know about the CPP file (Foo/extra.cpp).

Case 3

• lib_ldf_mode = deep
• Foo/extra.cpp depends on Bar library (contains #include <bar.h>)
• #include <foo.h> is located in one of the project source files

Firstly, LDF finds the Foo library, then it parses all sources from the Foo library and finds Foo/extra.cpp that depends on #include <bar.h>. Secondly, it will parse all sources from the Bar library. This operation continues until all dependencies will not be parsed.

Compatibility Mode

Compatibility mode allows one to control strictness of Library Dependency Finder. If library contains one of manifest file (library.json, library.properties, module.json), then LDF check compatibility of this library with real build environment. Available compatibility modes:

off Does not check for compatibility (is not recommended)
soft [DEFAULT] Checks for the compatibility with framework from build environment
strict Checks for the compatibility with framework and platform from build environment.

This mode can be changed using lib_compat_mode option in “platformio.ini” (Project Configuration File). Default value is set to soft.

C/C++ Preprocessor conditional syntax

In spite of the fact that Library Dependency Finder is written in pure Python, it evaluates C/C++ Preprocessor conditional syntax (#ifdef, if, defined, else, and elif) without calling gcc -E. For example,

```
[env:myenv]
lib_ldf_mode = chain+
bld_flags = -D MY_PROJECT_VERSION=13
```

mylib.h
#ifdef MY_PROJECT_VERSION
// include common file for the project
#include "my_common.h"
#endif

#if MY_PROJECT_VERSION < 10
// this include will be ignored because does not satisfy condition above
#include "my_old.h"
#endif

### 1.9.3 library.json

library.json is a manifest file of a library package. It allows developers to keep a project in its own structure and define:

- compatible frameworks and platforms
- external dependencies
- advanced build settings.

A data in library.json should be represented in JSON-style via associative array (name/value pairs). An order doesn’t matter. The allowable fields (names from pairs) are described below.

You can validate library.json manifest file using `pio package pack` command.

#### Fields

- `name`
- `version`
- `description`
- `keywords`
- `repository`
- `authors`
- `license`
- `homepage`
- `export`
  - `include`
  - `exclude`
- `frameworks`
- `platforms`
- `dependencies`
- `examples`
- `build`
  - `flags`
name

**Required** | **Type:** String | **Max. Length:** 50

A name of a library.

- Must be unique
- Should be slug style for simplicity, consistency, and compatibility. Example: *HelloWorld*
- Can contain a-z, digits, and dashes (but not start/end with them)
- Consecutive dashes and [:;/,@<>] chars are not allowed.

version

**Required** | **Type:** String | **Max. Length:** 20

A version of a current library source code. Can contain a-z, digits, dots or dash and should be Semantic Versioning compatible.

Example:

```json
"name": "Bar",
"version": "1.0.0",
"repository": {
   "type": "git",
   "url": "https://github.com/foo/bar.git"
}
```

description

**Required** | **Type:** String | **Max. Length:** 255

The field helps users to identify and search for your library with a brief description. Describe the hardware devices (sensors, boards and etc.) which are suitable with it.
keywords

**Required** | Type: String or Array | Max. Length: 255

Used for search by keyword. Helps to make your library easier to discover without people needing to know its name. The keyword should be lowercased, can contain a-z, digits and dash (but not start/end with them). A list from the keywords can be specified with separator , or declared as Array.

repository

**Optional** | Type: Object

The repository in which the source code can be found. The field consists of the next items:

- **type** the only “git”, “hg” or “svn” are supported
- **url**
- **branch** if is not specified, default branch will be used. This field will be ignored if tag/release exists with the value of version.

Example:

```json
"repository": {
  "type": "git",
  "url": "https://github.com/foo/bar.git"
}
```

authors

**Optional** | Type: Object or Array

An author contact information

- **name** Full name (**Required**)
- **email**
- **url** An author’s contact page
- **maintainer** Specify “maintainer” status

Examples:

```json
"authors": {
  "name": "John Smith",
  "email": "me@john-smith.com",
  "url": "https://www.john-smith/contact"
}
```

```json
..."authors": {

  "name": "John Smith",
  "email": "me@john-smith.com",
}
```

(continues on next page)


Note: If authors field is not defined, PlatformIO will try to fetch data from VCS provider (Github, Gitlab, etc) if repository is declared.

**license**

*Optional* | *Type:* String

A SPDX license ID of the library. You can check the full list of SPDX license IDs (see “Identifier” column).

"license": "Apache-2.0"

**homepage**

*Optional* | *Type:* String | Max. Length: 255

Home page of a library (if is different from repository url).

**export**

*Optional* | *Type:* Object

This option is useful if you need to exclude extra data (test code, docs, images, PDFs, etc). It allows one to reduce the size of the final archive.

To check which files will be included in the final packages, please use `pio package pack` command.

Possible options:

- *include*
- *exclude*

**include**

*Optional* | *Type:* Array | *Glob Pattern*

Export only files that matched declared patterns.

**Pattern Meaning**
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>matches everything</td>
</tr>
<tr>
<td>?</td>
<td>matches any single character</td>
</tr>
<tr>
<td>[seq]</td>
<td>matches any character in seq</td>
</tr>
<tr>
<td>![seq]</td>
<td>matches any character not in seq</td>
</tr>
</tbody>
</table>

Example:

```
"export": {
    "include": [
        "dir/*/.[ch]pp",
        "dir/examples/*",
        "*/*/.h"
    ]
}
```

**exclude**

*Optional* | Type: *Array* | *Glob Pattern*
Exclude the directories and files which match with *exclude* patterns.

**frameworks**

*Optional* | Type: *String* or *Array*
A list with compatible frameworks. The available framework names are defined in the *Frameworks* section.
Example:

```
"frameworks": ["espidf", "freertos"]
```

If the library is compatible with the all frameworks, then do not declare this field or you use * symbol:

```
"frameworks": "*
```

**platforms**

*Optional* | Type: *String* or *Array*
A list with compatible development platforms. The available platform names are defined in *Development Platforms* section.
Example:

```
"frameworks": ["atmelavr", "espressif8266"]
```

If the library is compatible with the all platforms, then do not declare this field or use * symbol:

```
"platforms": "*
```
dependencies

Optional | Type: Array or Object

A list of dependent libraries. They will be installed automatically with `pio lib install` command.

Allowed requirements for dependent library:

- `owner` | Type: String – an owner name (username) from the PlatformIO Registry
- `name` | Type: String – library name
- `version` | Type: String – version or version range in SemVer format
- `frameworks` | Type: String or Array – project compatible Frameworks
- `platforms` | Type: String or Array – project compatible Development Platforms

The version supports Semantic Versioning (`<major>.<minor>.<patch>`) and can take any of the following forms:

- `1.2.3` - an exact version number. Use only this exact version
- `^1.2.3` - any compatible version (exact version for `1.x.x` versions
- `~1.2.3` - any version with the same major and minor versions, and an equal or greater patch version
- `>1.2.3` - any version greater than `1.2.3`.
- `>=`, `<`, and `<=` are also possible
- `>0.1.0,!=0.2.0,<0.3.0` - any version greater than `0.1.0`, not equal to `0.2.0` and less than `0.3.0`

The rest possible values including VCS repository URLs are documented in `pio lib install` command.

Example:

```
"dependencies":
  [
    {
      "owner": "bblanchon",
      "name": "ArduinoJson",
      "version": "^6.16.1"
    },
    {
      "owner": "me-no-dev",
      "name": "AsyncTCP",
      "version": "*",
      "platforms": ["espressif32"]
    },
    {
      "name": "external-repo",
      "version": "https://github.com/user/package.git@1.2.3"
    },
    {
      "name": "external-zip",
      "version": "https://github.com/me-no-dev/AsyncTCP/archive/master.zip"
    }
  ]
```

A short definition of dependencies is allowed:

```
"dependencies":
  {
    "bblanchon/ArduinoJson": "^6.16.1",
  }
```

(continues on next page)
examples

Optional | Type: Array | Glob Pattern

A list of example patterns. This field is predefined with default value:

```json
"examples": [
  {
    "name": "Hello",
    "base": "examples/world",
    "files": [
      "platformio.ini",
      "include/world.h",
      "src/world.c",
      "README",
      "extra.py"
    ]
  },
  {
    "name": "Blink",
    "base": "examples/blink",
    "files": ["blink.cpp", "blink.h"]
  }
]
```

build

Optional | Type: Object

Specify advanced settings, options and flags for the build system. Possible options:

- flags
- unflags
- includeDir
- srcDir
- srcFilter
- extraScript
- libArchive
- libLDFMode
- libCompatMode
flags

Optional | Type: String or Array
Extra flags to control preprocessing, compilation, assembly and linking processes. More details build_flags.

unflags

Optional | Type: String or Array
Remove base/initial flags which were set by development platform. More details build_unflags.

includeDir

Optional | Type: String
Custom location of library header files. A default value is include and means that folder is located in the root of a library.

srcDir

Optional | Type: String
Custom location of library source code. A default value is src and means that folder is located in the root of a library.

srcFilter

Optional | Type: String or Array
Specify which source files should be included/excluded from build process. The path in filter should be relative to the srcDir option of a library.
See syntax in src_filter.
Please note that you can generate source filter “on-the-fly” using extraScript (see below)

extraScript

Optional | Type: String
Launch extra script before build process. More details extra_scripts.

Example (HAL-based library)
This example demonstrates how to build HAL-dependent source files and exclude other source files from a build process.
Project structure

```
lib
    README
    SomeLib
        extra_script.py
        hal
```
(continues on next page)
platformio.ini

[env:foo]
platform = native
build_flags = -DHAL=foo

[env:bar]
platform = native
build_flags = -DHAL=bar

library.json

{
    "name": "SomeLib",
    "version": "0.0.0",
    "build": {
        "extraScript": "extra_script.py"
    }
}

extra_script.py

Import('env')
from os.path import join, realpath

# private library flags
for item in env.get("CPPDEFINES", []):
    if isinstance(item, tuple) and item[0] == "HAL":
        env.Append(CPPPATH=[realpath(join("hal", item[1]))])
        env.Replace(SRC_FILTER=["+<*>", "-<hal>", "+<%s>" % join("hal", item[1])])
        break

# pass flags to a global build environment (for all libraries, etc)
global_env = DefaultEnvironment()
global_env.Append(
    CPPDEFINES=[
        ("MQTT_MAX_PACKET_SIZE", 512),
        "ARDUINOJSON_ENABLE_STD_STRING",
        ("BUFFER_LENGTH", 32)
    ]
)
libArchive

*Optional | Type: Boolean*

Create an archive (*.a, static library) from the object files and link it into a firmware (program). This is default behavior of PlatformIO Build System ("libArchive": true).

Setting "libArchive": false will instruct PlatformIO Build System to link object files directly (in-line). This could be useful if you need to override weak symbols defined in framework or other libraries.

You can disable library archiving globally using *lib_archive* option in “platformio.ini” (Project Configuration File).

libLDFMode

*Optional | Type: String*

Specify Library Dependency Finder Mode. See Dependency Finder Mode for details.

libCompatMode

*Optional | Type: String*

Specify Library Compatibility Mode. See Compatibility Mode for details.

Examples

1. Custom macros/defines

```json
"build": {
    "flags": "-D MYLIB_REV=1.2.3 -DRELEASE"
}
```

2. Extra includes for C preprocessor

```json
"build": {
    "flags": [
        "-I inc",
        "-I inc/target_x13"
    ]
}
```

3. Force to use C99 standard instead of C11

```json
"build": {
    "unflags": "-std=gnu++11",
    "flags": "-std=c99"
}
```

4. Build source files (.c, .cpp, .h) at the top level of the library

```json
"build": {
    "srcFilter": [
        "+<*.c>",
        "+<*.cpp>",
        "+<*.h>"
    ]
}
```

(continues on next page)
5. Extend PlatformIO Build System with own extra script

```json
"build": {
    "extraScript": "generate_headers.py"
}
```

generate_headers.py

```python
import ('env')
# print(env.Dump())
env.Append(
    CPPDEFINES=["HELLO=\"WORLD\", "TAG=1.2.3", "DEBUG"],
    CPPPATH=["inc", "inc/devices"]
)

# some python code that generates header files "on-the-fly"
```

### 1.9.4 Creating Library

Managing components between the projects is a historical issue. A common code is duplicated between different projects that lead to project complexity. A good practice is to organize interdependent components as the separate libraries where other projects can depend on them.

PlatformIO has a built-in *Library Management* where developers can declare project dependencies and PlatformIO will automatically manage them (install, build, update). It doesn’t have any requirements for a library source code structure. The only requirement is a library manifest file - `library.json`, `library.properties`, or `module.json`. It must be located in the root of a library.

We highly recommend using `library.json` for better compatibility and avoiding any issues.

#### Contents

- **Structure**
- **Manifest**
- **Publishing**
- **Examples**

#### Structure

We recommend to use `src` folder for your C/C++ source files and `include` folder for your headers. You can also have nested sub-folders in `src` or `include`.

Example

```
    ├── examples
    │     └── echo
    └── include
```

(continues on next page)
A library package must contain a manifest. We recommend using `library.json`.

**Example**

```json
{
  "name": "HelloWorld",
  "version": "1.0.0",
  "description": "A "Hello world" program is a computer program that outputs "Hello World" (or some variant) on a display device",
  "keywords": "planet, happiness, people",
  "repository": {
    "type": "git",
    "url": "https://github.com/username/hello-world.git"
  },
  "authors": [
    {
      "name": "John Smith",
      "email": "me@john-smith.com",
      "url": "https://www.john-smith/contact"
    },
    {
      "name": "Andrew Smith",
      "email": "me@andrew-smith.com",
      "url": "https://www.andrew-smith/contact",
      "maintainer": true
    }
  ],
  "license": "MIT",
  "homepage": "https://www.helloworld.org/",
  "dependencies": {
    "ownername/print": "~1.3.0"
  },
  "frameworks": "*",
  "platforms": "*"
}
```

**Publishing**

You can publish a library to the PlatformIO Registry using `pio package publish` command. Every time when you modify a source code of a library you will need to increment the “version” field in `library.json` manifest and re-publish again.

If the published library has an issue and you would like to remove it from the PlatformIO Registry, please use `pio package unpublish` command.
Examples

See the published libraries in PlatformIO Registry.

1.10 Development Platforms

The PlatformIO ecosystem has a decentralized architecture, allowing development for a range of development platforms. A development platform (or just “platform” for short) is usually a particular microcontroller or processor architecture that PlatformIO projects can be compiled to run on. (A few platforms, for example Teensy, use different target architectures for different boards.)

Each of the three supported host systems Mac OS X, Linux and Windows support compiling for all platforms listed below. Some platforms are also supported under ARM Linux hosts such as Raspberry Pi. For each development platform, PlatformIO defines:

- The PlatformIO Build System build scripts for the supported frameworks and SDKs
- Pre-configured presets for embedded circuit boards
- Pre-compiled toolchains and related tools for the architecture(s)

Each project must specify the platform name using the platform option in “platformio.ini” (Project Configuration File). A specific platform version can optionally be specified as well. As embedded boards are equipped with a particular microcontroller, each embedded board specifies what development platform it uses and this can not be changed.

If a new board uses an architecture not in this list, a custom development platform can be created; see Custom Development Platforms.

1.10.1 Embedded

Aceinna IMU

Configuration platform = aceinna imu

Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

For more detailed information please visit vendor site.

Examples

Examples are listed from Aceinna IMU development platform repository:
- OpenIMU300RI
- OpenIMU330BI
- OpenRTK330LI
- OpenIMU300ZI

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools

**Tools & Debug Probes**

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

**On-Board Debug Tools**

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aceinna Low Cost RTK</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
</tbody>
</table>

**External Debug Tools**

Boards listed below are compatible with *Debugging* but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aceinna OpenIMU 300</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300ZA</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330</td>
<td>STM32L431CB</td>
<td>80MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330ZA</td>
<td>STM32F469IG</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>Aceinna OpenRTK330L</td>
<td>STM32F469IG</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
</tbody>
</table>
Stable and upstream versions

You can switch between stable releases of Aceinna IMU development platform and the latest upstream version using `platform` option in “`platformio.ini`” (Project Configuration File) as described below.

### Stable

```ini
; Latest stable version
[env:latest_stable]
platform = aceinna_imu
board = ...

; Custom stable version
[env:custom_stable]
platform = aceinna_imu@x.y.z
board = ...
```

### Upstream

```ini
[env:upstream_develop]
platform = https://github.com/aceinna/platform-aceinna_imu.git
board = ...
```

### Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

**Warning:** Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

**Windows Users:**

Please check that you have a correctly installed USB driver from board manufacturer

### Boards

**Note:**

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
For more detailed board information please scroll the tables below by horizontally.

### Aceinna

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aceinna Low Cost RTK</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300ZA</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330</td>
<td>External</td>
<td>STM32L431CB</td>
<td>80MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330ZA</td>
<td>External</td>
<td>STM32F469IG</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>Aceinna OpenRTK330L</td>
<td>External</td>
<td>STM32F469IG</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
</tbody>
</table>

### ASR Microelectronics ASR605x

**Configuration**  
`platform = asrmicro650x`

ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

For more detailed information please visit [vendor site](#).

**Contents**

- Configuration
- Examples
- Stable and upstream versions
- Packages
- Frameworks
- Boards

### Configuration

- **LoRaWAN**

**LoRaWAN**

LoRaWAN protocol can be configured in "platformio.ini" (`Project Configuration File`) using the following syntax:

```
board_build.arduino.lorawan.*
```

where `*` is an option from the following list:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Possible values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>Device class</td>
<td>CLASS_A, CLASS_C</td>
<td>CLASS_A</td>
</tr>
<tr>
<td>netmode</td>
<td>Activation method</td>
<td>OTAA, ABP</td>
<td>OTAA</td>
</tr>
<tr>
<td>adr</td>
<td>Adaptive Data Rate</td>
<td>ON, OFF</td>
<td>ON</td>
</tr>
<tr>
<td>uplinkmode</td>
<td>Uplink confirmed/unconfirmed messages</td>
<td>CONFIRMED, UNCONFIRMED</td>
<td>CONFIRMED</td>
</tr>
<tr>
<td>net_reserve</td>
<td>Don’t rejoin after reset</td>
<td>ON, OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>at_support</td>
<td>AT commands support</td>
<td>ON, OFF</td>
<td>ON</td>
</tr>
<tr>
<td>rgb</td>
<td>RGB light for LoRaWAN status</td>
<td>ACTIVE, DEACTIVE</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>debug_level</td>
<td>Print LoRaWAN relevant messages print to serial port</td>
<td>NONE, FREQ (Sending/receiving frequency), FREQ_AND_DIO (Sending/receiving frequency and DIO pin interrupt information)</td>
<td>NONE</td>
</tr>
</tbody>
</table>

Example

```Ini
[env:cubecell_board]
platform = asrmicro650x
framework = arduino
board = cubecell_board
board_build.arduino.lorawan.region = EU433
board_build.arduino.lorawan.adr = OFF
board_build.arduino.lorawan.debug_level = FREQ_AND_DIO
```

More information about LoRaWAN configuration can be found in the official CubeCell documentation.

Examples

Examples are listed from ASR Microelectronics ASR605x development platform repository:

- arduino-blink
- arduino-rgb
- arduino-adc
- LoRa
- arduino-lowpower

Stable and upstream versions

You can switch between stable releases of ASR Microelectronics ASR605x development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.
Stable

```plaintext
; Latest stable version
[env:latest_stable]
platform = asrmicro650x
board = ...

; Custom stable version
[env:custom_stable]
platform = asrmicro650x@x.y.z
board = ...
```

Upstream

```plaintext
[env:upstream_develop]
platform = https://github.com/HelTecAutomation/platform-asrmicro650x.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinoasrmicro650x</td>
<td>Arduino Wiring-based Framework for ASR Microelectronics ASR650x (Heltec core)</td>
</tr>
<tr>
<td>tool-cubecellelftool</td>
<td>CubeCell ELF tool</td>
</tr>
<tr>
<td>tool-cubecellflash</td>
<td>CubeCell Flash tool</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article `Enable serial port on Raspberry Pi`.

Windows Users:
Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Boards
Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer.
- For more detailed board information please scroll the tables below by horizontally.

### Heltec

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec CubeCell Capsule Solar Sensor (HTCC-AC02)</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-1/2AA Node (HTCC-AB02A)</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Board (HTCC-AB01)</td>
<td>No</td>
<td>ASR6501</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Board Plus (HTCC-AB02)</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Capsule (HTCC-AC01)</td>
<td>No</td>
<td>ASR6501</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-GPS (HTCC-AB02S)</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Module (HTCC-AM01)</td>
<td>No</td>
<td>ASR6501</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Module Plus (HTCC-AM02)</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Atmel AVR

#### Configuration

Configuration `platform = atmelavr`

Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

For more detailed information please visit vendor site.

### Contents

- Configuration
- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

### Configuration

- Upload using Programmer
- Upload EEPROM data
• **Fuses programming**
  – Custom fuses
  – MiniCore, MegaCore, MightyCore, MajorCore and MicroCore
  – Overriding default fuses command

• **Bootloader programming**
  – Custom bootloader
  – Overriding default bootloader command

### Upload using Programmer

In the case of external programmers, it’s easy to brick a board simply by specifying incorrect upload flags. It’s highly recommended to use the `upload_command` option that gives the full control over flags used for uploading.

**Note:** The list of supported programmers available in avrdude is accessible via the `avrdude -c ?` command

Configuration for the programmers:

• **AVRISP**

```python
[env:program_via_AVRISP]
platform = atmelavr
framework = arduino
upload_protocol = custom
upload_port = SERIAL_PORT_HERE
upload_speed = 19200
upload_flags =
  -C
  ; use "tool-avrdude-megaavr" for the atmelmegaavr platform
$PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
-p
$BOARD_MCU
-p
$UPLOAD_PORT
-b
$UPLOAD_SPEED
-c
stk500v1
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i
```

• **AVRISP mkII**

```python
[env:program_via_AVRISP_mkII]
platform = atmelavr
framework = arduino
upload_protocol = custom
upload_port = usb
upload_flags =
  -C
  ; use "tool-avrdude-megaavr" for the atmelmegaavr platform
$PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
```

(continues on next page)
• **USBtinyISP**

```
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = usbtiny

[env:program_via_USBtinyISP]
platform = atmelavr
framework = arduino
upload_protocol = custom
upload_flags =
  -C
  ; use "tool-avrdude-megaavr" for the atmelmegaavr platform
  $PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
  -P
  $BOARD_MCU
  -c
  usbtiny
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i
```

• **Arduino as ISP**

```
[env:program_via_ArduinoISP]
platform = atmelavr
framework = arduino
upload_protocol = custom
upload_port = $SERIAL_PORT_HERE
upload_speed = 19200
upload_flags =
  -C
  ; use "tool-avrdude-megaavr" for the atmelmegaavr platform
  $PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
  -P
  $BOARD_MCU
  -P
  $UPLOAD_PORT
  -b
  $UPLOAD_SPEED
  -c
  stk500v1
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i
```

• **USBasp**

```
[env:program_via_USBasp]
platform = atmelavr
framework = arduino
upload_protocol = custom
```

(continues on next page)
upload_port = usb
upload_flags =
  -C
  ; use "tool-avrdude-megaavr" for the atmelmegaavr platform
  $PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
  -p $BOARD_MCU
  -P $UPLOAD_PORT
  -c usbasp
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i

• Parallel Programmer

[env:program_via_PP]
platform = atmelavr
framework = arduino
upload_protocol = custom
upload_flags =
  -C
  ; use "tool-avrdude-megaavr" for the atmelmegaavr platform
  $PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
  -p $BOARD_MCU
  -c dapa
  -F
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i

• Bus Pirate as ISP

[env:program_via_BP]
platform = atmelavr
framework = arduino
upload_protocol = custom
upload_port = SERIAL_PORT_HERE
upload_speed = 115200
upload_flags =
  -C
  ; use "tool-avrdude-megaavr" for the atmelmegaavr platform
  $PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf
  -p $BOARD_MCU
  -P $UPLOAD_PORT
  -b $UPLOAD_SPEED
  -c buspirate
upload_command = avrdude $UPLOAD_FLAGS -U flash:w:$SOURCE:i
Upload EEPROM data

To upload EEPROM data (from EEMEM directive) you need to use uploadeep target instead upload for pio run --target command. For example, pio run -t uploadeep.

Fuses programming

PlatformIO has a built-in target called fuses for setting fuse bits. The default fuse bits are predefined in the board manifest file in the fuses section. For example, fuses section for Arduino Uno board. To set fuse bits you need to use target fuses with pio run --target command.

Custom fuses

Custom fuse values and upload flags (based on upload protocol) should be specified in “platformio.ini” (Project Configuration File). The lfuse and hfuse bits are mandatory, efuse is optional and not supported by all targets. An example of setting custom fuses for uno board:

```
[env:custom_fuses]
platform = atmelavr
framework = arduino
board = uno
upload_protocol = stk500v1
upload_speed = 19200
board_fuses.lfuse = 0xAA
board_fuses.hfuse = 0xBB
board_fuses.efuse = 0xCC
upload_flags =
   -PCOM15
   -b$UPLOAD_SPEED
   -e
```

MiniCore, MegaCore, MightyCore, MajorCore and MicroCore

MiniCore, MegaCore, MightyCore, MajorCore and MicroCore support dynamic fuses generation. Generated values are based on the following parameters:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>board_build.f_cpu</td>
<td>Specifies the clock frequencies in Hz. Used to determine what oscillator option to choose. A capital L has to be added to the end of the frequency number.</td>
<td>16000000L</td>
</tr>
<tr>
<td>board_hardware.oscillator</td>
<td>Specifies which oscillator is used internal or external. Internal oscillator only works with f_cpu values 8000000L and 1000000L</td>
<td>external</td>
</tr>
<tr>
<td>board_hardware.uart</td>
<td>Specifies the hardware UART port used for serial upload. can be uart0, uart1, uart2 or uart3 depending on the target. Use no_bootloader if you're not using a bootloader for serial upload.</td>
<td>uart0</td>
</tr>
<tr>
<td>board_hardware.bod</td>
<td>Specifies the hardware brown-out detection. Use disabled to disable brown-out detection.</td>
<td>2.7v</td>
</tr>
<tr>
<td>board_hardware.eesave</td>
<td>Specifies if the EEPROM memory should be retained when uploading using a programmer. Use no to disable</td>
<td>yes</td>
</tr>
<tr>
<td>board_hardware.ckout</td>
<td>Enables system clock output on targets that have this feature. The system clock will be output on a dedicated output pin. See the target datasheet for more information. Use Yes to enable</td>
<td>no</td>
</tr>
<tr>
<td>board_hardware.jtagen</td>
<td>Enables the JTAG programming and debugging interface for targets that supports JTAG. Use yes to enable</td>
<td>no</td>
</tr>
<tr>
<td>board_hardware.cfd</td>
<td>Enables clock failure detection. Note that this feature is only available on ATmega324PB and ATmega328PB. Use yes to enable CFD</td>
<td>no</td>
</tr>
</tbody>
</table>

Valid BOD values:

<table>
<thead>
<tr>
<th>ATmega8, ATmega8515, ATmega8535/16/32, ATmega64/128</th>
<th>AT90CAN32/64/128</th>
<th>Other targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0v</td>
<td>4.1v</td>
<td>4.3v</td>
</tr>
<tr>
<td>2.7v</td>
<td>4.0v</td>
<td>2.7v</td>
</tr>
<tr>
<td>disabled</td>
<td>3.9v</td>
<td>1.8v</td>
</tr>
<tr>
<td></td>
<td>3.8v</td>
<td>disabled</td>
</tr>
<tr>
<td></td>
<td>2.7v</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.6v</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.5v</td>
<td></td>
</tr>
<tr>
<td></td>
<td>disabled</td>
<td></td>
</tr>
</tbody>
</table>

Hardware configuration example:

```yaml
[env:custom_fuses]
platform = atmelavr
framework = arduino
board = ATmega32

board_build.f_cpu = 1000000L
board_hardware.uart = uart0
board_hardware.oscillator = internal
board_hardware.bod = 2.7v
board_hardware.eesave = no
upload_protocol = usbasp
```

(continues on next page)
Overriding default fuses command

PlatformIO splits the command for programming fuses in the following overridable parts:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUSESUPLOADER</td>
<td>The tool used for setting fuses</td>
<td>By default <code>avrdude</code> is used</td>
</tr>
<tr>
<td>FUSESUPLOADERFLAGS</td>
<td>General command-line options that control uploader’s behavior</td>
<td>-D, -V, -P COM1, -C <code>atmelice_isp</code>, -b 115200</td>
</tr>
<tr>
<td>FUSESFLAGS</td>
<td>A list of flags specific to fuses settings</td>
<td>-Ulock:w:0x2F:m, -Uefuse:w:0xCB:m, -Ulfuse:w:0xFF:m</td>
</tr>
<tr>
<td>SETFUSESCMD</td>
<td>Variable that holds the final command compiled from variables above</td>
<td><code>$FUSESUPLOADER</code></td>
</tr>
</tbody>
</table>

If for any reason default parameters are not suitable for your project, you can override the entire upload command or any particular part of that command using an extra script, for example, you can override only fuses values:

```python
Import("env")
env.Replace(
    FUSESFLAGS=[
        "-Uhfuse:w:0xAA:m",
        "-Uefuse:w:0xBB:m",
        "-Ulfuse:w:0xCC:m",
        "-Ulock:w:0xDD:m"
    ]
)
```

Or override a specific uploader flag:

```python
Import("env")
env.Append(
    FUSESUPLOADERFLAGS=[
        "-V",
        "-D"
    ]
)
```

It’s also possible to completely override the entire upload command:

```python
import("env")
env.Replace(
    FUSESUPLOADERFLAGS=[
        # use "tool-avrdude-megaavr" for the atmelmegaavr platform
        
        "-C", "$PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf",
        "-p", "$BOARD_MCU",
        "-c", "atmelice_isp",
    ]
)
```
Bootloader programming

PlatformIO has a built-in target called `bootloader` for flashing bootloaders. The default bootloader image and corresponding fuse bits are predefined in the board manifest file in the `bootloader` section, for example, Arduino Uno. To upload a bootloader image you need to use target `bootloader` with `pio run --target` command.

Custom bootloader

Custom bootloader and accompanying fuses should be specified in the `platformio.ini` (Project Configuration File). If `lock_bits` and `unlock_bits` are not set then the default values 0x0F and 0x3F are used accordingly. An example of setting custom bootloader for `uno` board:

```
[env:uno]
platform = atmelavr
framework = arduino
board = uno

board_bootloader.file = /path/to/custom/bootloader.hex
board_bootloader.1fuse = 0xFF
board_bootloader.hfuse = 0xDE
board_bootloader.efuse = 0xFD
board_bootloader.lock_bits = 0x0F
board_bootloader.unlock_bits = 0x3F
```

MiniCore, MegaCore, MightyCore and MajorCore have a wide variety of precompiled bootloaders. Bootloader binaries are dynamically selected according to the hardware parameters `f_cpu`, `oscillator`, `uart` and `upload_speed`. For a complete table with all available baud rates, see the Optiboot flash repo. Here is a table with recommended baud rates for different clock frequencies:

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Oscillator type</th>
<th>Recommended upload speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000000L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>18432000L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>16000000L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>14745600L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>12000000L</td>
<td>external</td>
<td>57600</td>
</tr>
<tr>
<td>11059200L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>8000000L</td>
<td>external/internal</td>
<td>57600/38400</td>
</tr>
<tr>
<td>7372800L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>4000000L</td>
<td>external</td>
<td>9600</td>
</tr>
<tr>
<td>3686400L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>2000000L</td>
<td>external</td>
<td>9600</td>
</tr>
<tr>
<td>1843200L</td>
<td>external</td>
<td>115200</td>
</tr>
<tr>
<td>1000000L</td>
<td>external/internal</td>
<td>9600</td>
</tr>
</tbody>
</table>
Overriding default bootloader command

PlatformIO splits the command for programming bootloader in the following overridable parts:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOTUPLOADER</td>
<td>The tool used for programming bootloader image</td>
<td>By default avrdude is used</td>
</tr>
<tr>
<td>BOOTUPLOADERS</td>
<td>General command-line options that control uploader’s behavior</td>
<td>-D, -V, -P COM1, -C atmelice_isp, -b 115200</td>
</tr>
<tr>
<td>BOOTFLAGS</td>
<td>A list of flags specific to bootloader settings</td>
<td>-Uflash:w:/path/to/ bootlader_image:i, -Ulock:w:0x2F:m</td>
</tr>
<tr>
<td>UPLOADBOOTCMD</td>
<td>Variable that holds the final command compiled from variables above</td>
<td>$BOOTUPLOADER $BOOTUPLOADERS $UPLOAD_FLAGS $BOOTFLAGS</td>
</tr>
</tbody>
</table>

If for any reason default parameters are not suitable for your project, you can override the entire upload command or any particular part of that command using an extra script, for example, you can override only fuses values:

```python
Import("env")
bootloader_path = "~/path/to/custom/bootloader.hex"
env.Replace(
    BOOTFLAGS=[
        "-Uflash:w:%s:i" % bootloader_path,
        "-Ulock:w:0xFF:m"
    ]
)
```

Or override a specific uploader flag:

```python
Import("env")
env.Append(
    BOOTUPLOADERSFLAGS=[
        "-e", "-p", "~/dev/cu.usbserial-1414302"
    ]
)
```

It's also possible to completely override the entire upload command:

```python
Import("env")
env.Replace(
    BOOTUPLOADERSFLAGS=[
        # use "tool-avrdude-megaavr" for the atmelmegaavr platform
        "-C", "$PROJECT_PACKAGES_DIR/tool-avrdude/avrdude.conf",
        "-p", "$BOARD_MCU",
        "-c", "atmelice_isp"
    ],
    UPLOADBOOTCMD="avrdude $BOOTUPLOADERSFLAGS -Ulock:w:0x0F:m",
)
```
Examples

Examples are listed from Atmel AVR development platform repository:

• arduino-blink
• simba-blink
• native-blink
• arduino-own-src_dir
• engduino-magnetometer
• digitstump-mouse
• arduino-internal-libs
• arduino-external-libs

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>HAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega128/A</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega1280</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1281</td>
<td>ATMEGA1281</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1284</td>
<td>ATMEGA1284</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega1284P</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega16</td>
<td>ATMEGA16</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164P/PA</td>
<td>ATMEGA164P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega168/A</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168P/PA</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>256KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega324A</td>
<td>ATMEGA324A</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324P</td>
<td>ATMEGA324P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PA</td>
<td>ATMEGA324PA</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328</td>
<td>ATMEGA328</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328P/PA</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48/A</td>
<td>ATMEGA48</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega48P/PA</td>
<td>ATMEGA48P</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega644P/PA</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>64KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega8/A</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88/A</td>
<td>ATMEGA88</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88P/PA</td>
<td>ATMEGA88P</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny13</td>
<td>ATTINY13</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>ATtiny13A</td>
<td>ATTINY13A</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>Adafruit Bluefruit Micro</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Feather 328P</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Feather 32u4</td>
<td>ATMEGA328U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>ATMEGA328U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/8MHz</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/16MHz</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (FTDI)</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (USB)</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (USB)</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Trinket 3V/8MHz</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit Trinket 5V/16MHz</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Alorium Hinj</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Anarduinu MiniWireless</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduboy</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduboy DevKit</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino BT ATmega168</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega168</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega328</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Explora</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Industrial 101</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Leonardo ETH</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino LilyPad USB</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>124KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega168</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega8</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega168</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Yun Mini</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>BQ ZUM BT-328</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>BitWizard Raspduino</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Controllino Maxi</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Digestpark USB</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>5.87KB</td>
<td>512B</td>
</tr>
<tr>
<td>Engduino 3</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>EnviroDIY Mayfly</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>FYSETC F6 V1.3</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Generic ATtiny2313</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>ATTINY25</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny4313</td>
<td>ATTINY4313</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny44</td>
<td>ATTINY44</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny45</td>
<td>ATTINY45</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny84</td>
<td>ATTINY84</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny85</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>LightBlue Bean</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LightUp</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Linino One</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>LinkIt Smart 7688 Duo</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>LoRa32u4II (868-915MHz)</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>LowPowerLab MightyHat</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8Mhz)</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab MoteinoMEGA</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core (Atmega168@16M,5V)</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega168@8M,3.3V)</td>
<td>ATMEGA168P</td>
<td>8MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328@16M,5V)</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328@8M,3.3V)</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core USB (Atmega32U4@16M,5V)</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega1284P@16M,5V)</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega1284P@8M,3.3V)</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@16M,5V)</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@8M,3.3V)</td>
<td>ATMEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>OpenEnergyMonitor emonPi</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Original Prusa i3 MK3 Multi Material 2.0 Upgrade</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>PanStamp AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Pololu A-Star 32U4</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Prusa RAMBo</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Quirkbot</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/16MHz (overclock)</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/8MHz</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RepRap RAMBo</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SODAQ GaLoRa</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Mbili</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Moja</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SODAQ Ndogo</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Tatu</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (16MHz)</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>ATMEGA644</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>ATMEGA644</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (16 MHz)</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>ATMEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Seeedduino</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun ATmega128RFA1 Dev Board</td>
<td>ATMEGA128RFA1</td>
<td>16MHz</td>
<td>16KB</td>
<td>124KB</td>
</tr>
<tr>
<td>SparkFun Digital Sandbox</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Fio V3 3.3V/8MHz</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Makey Makey</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3.3V/8MHz</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 5V/16MHz</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro Mini 3.3V</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 3.3V/8MHz</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 5V/16MHz</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Qduino Mini</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Spellfoundry Sleepy Pi 2</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Talk2 Whisper Node</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>The Things Uno</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>TinyCircuits TinyDuino Processor Board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>USBasp stick</td>
<td>ATMEGA8</td>
<td>12MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V2</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>120.00KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V3</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ftDuino</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>nicai-systems BOB3 coding bot</td>
<td>ATMEGA88</td>
<td>8MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBO 2 robot</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot with Tuning Kit</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot with Tuning Kit</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ubIQio Ardhat</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Atmel AVR development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

`; Latest stable version
[env:latest_stable]
platform = atmelavr
board = ...

`; Custom stable version
[env:custom_stable]
platform = atmelavr@x.y.z
board = ...

**Upstream**

[env:upstream_develop]
platform = https://github.com/platformio/platform-atmelavr.git
board = ...
### Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduino-avr</td>
<td>The official Arduino Wiring-based Framework for Microchip AVR microcontrollers</td>
</tr>
<tr>
<td>framework-arduino-avr-attiny</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (ATTiny Core)</td>
</tr>
<tr>
<td>framework-arduino-avr-bean</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (Bean Core)</td>
</tr>
<tr>
<td>framework-arduino-avr-core13</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (Core13)</td>
</tr>
<tr>
<td>framework-arduino-avr-digistump</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (Digistump Core)</td>
</tr>
<tr>
<td>framework-arduino-avr-dwenguino</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (Dwenguino Core)</td>
</tr>
<tr>
<td>framework-arduino-avr-majorcore</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (MajorCore)</td>
</tr>
<tr>
<td>framework-arduino-avr-megacore</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (MegaCore)</td>
</tr>
<tr>
<td>framework-arduino-avr-microcore</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (MicroCore)</td>
</tr>
<tr>
<td>framework-arduino-avr-mightycore</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (MightyCore)</td>
</tr>
<tr>
<td>framework-arduino-avr-minicore</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (MiniCore)</td>
</tr>
<tr>
<td>framework-arduino-avr-nicai</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (Nicai Core)</td>
</tr>
<tr>
<td>framework-arduino-avr-panstamp</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (Panstamp Core)</td>
</tr>
<tr>
<td>framework-arduino-avr-prusa_rambo</td>
<td>Arduino Wiring-based Framework for Microchip AVR microcontrollers (Prusa Rambo Core)</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba is an Embedded Programming Platform. It aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>tool-avrdude</td>
<td>AVRDUDE is a utility to download/upload/manipulate the ROM and EEPROM contents of AVR microcontrollers</td>
</tr>
<tr>
<td>tool-micronucleus</td>
<td>ATTiny usb bootloader with a strong emphasis on bootloader compactness</td>
</tr>
<tr>
<td>tool-simavr</td>
<td>simavr is a lean, mean and hackable AVR simulator</td>
</tr>
<tr>
<td>toolchain-atmelavr</td>
<td>GCC Toolchain for Microchip AVR microcontrollers</td>
</tr>
</tbody>
</table>

**Warning: Linux Users:**

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

**Windows Users:**

Please check that you have a correctly installed USB driver from board manufacturer
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer.
- For more detailed board information please scroll the tables below by horizontally.

Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Bluefruit Micro</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Feather 328P</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Feather 32u4</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/8MHz</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 5V/16MHz</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (FTDI)</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (USB)</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (USB)</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Trinket 3V/8MHz</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit Trinket 5V/16MHz</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
</tbody>
</table>

Alorium Technology

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alorium Hinj</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>
Anarduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anarduino MiniWireless</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Arduboy

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduboy</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>2KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduboy DevKit</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>2KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino BT ATmega168</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega168</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega328</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Espora</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Leonardo ETH</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino LilyPad USB</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>124KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega168</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega8</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega168</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8Hz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8Hz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Yun Mini</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>
### Atmel

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic ATtiny1634</td>
<td>No</td>
<td>ATTINY1634</td>
<td>8MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Generic ATtiny167</td>
<td>No</td>
<td>ATTINY167</td>
<td>8MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny2313</td>
<td>On-board</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>On-board</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>On-board</td>
<td>ATTINY25</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny261</td>
<td>No</td>
<td>ATTINY261</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny4313</td>
<td>On-board</td>
<td>ATTINY4313</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny43U</td>
<td>No</td>
<td>ATTINY43U</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny44</td>
<td>On-board</td>
<td>ATTINY44</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny441</td>
<td>No</td>
<td>ATTINY441</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny45</td>
<td>On-board</td>
<td>ATTINY45</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny461</td>
<td>No</td>
<td>ATTINY461</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny48</td>
<td>No</td>
<td>ATTINY48</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny828</td>
<td>No</td>
<td>ATTINY828</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny84</td>
<td>On-board</td>
<td>ATTINY84</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny841</td>
<td>No</td>
<td>ATTINY841</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny85</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny861</td>
<td>No</td>
<td>ATTINY861</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny87</td>
<td>No</td>
<td>ATTINY87</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny88</td>
<td>No</td>
<td>ATTINY88</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>USBasp stick</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>12MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
</tbody>
</table>

### BQ

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BQ ZUM BT-328</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### BSFrance

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoRa32u4II (868-915MHz)</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### BitWizard

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitWizard Raspduino</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>
### Controllino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllino Maxi</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Digistump

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digispark Pro</td>
<td>No</td>
<td>ATTINY167</td>
<td>16MHz</td>
<td>14.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digispark Pro (16 MHz) (64 byte buffer)</td>
<td>No</td>
<td>ATTINY167</td>
<td>16MHz</td>
<td>14.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digispark Pro (32 byte buffer)</td>
<td>No</td>
<td>ATTINY167</td>
<td>16MHz</td>
<td>14.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digispark USB</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>5.87KB</td>
<td>512B</td>
</tr>
</tbody>
</table>

### Dwengo

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dwenguino</td>
<td>No</td>
<td>AT90USB646</td>
<td>16MHz</td>
<td>60KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Elektor

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elektor Uno R4</td>
<td>No</td>
<td>ATMEGA328PB</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Engduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engduino 3</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### EnviroDIY

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnviroDIY Mayfly</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### FYSETC

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYSETC F6 V1.3</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>
### LightUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightUp</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### Linino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linino One</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### LowPowerLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowPowerLab MightyHat</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8Mhz)</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab MoteinoMEGA</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### MediaTek Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkIt Smart 7688 Duo</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### Microchip

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT90CAN128</td>
<td>No</td>
<td>AT90CAN128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>AT90CAN32</td>
<td>No</td>
<td>AT90CAN32</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>AT90CAN64</td>
<td>No</td>
<td>AT90CAN64</td>
<td>16MHz</td>
<td>64KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega128/A</td>
<td>On-board</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega1280</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1281</td>
<td>On-board</td>
<td>ATMEGA1281</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1284</td>
<td>On-board</td>
<td>ATMEGA1284</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega1284P</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega16</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega162</td>
<td>No</td>
<td>ATMEGA162</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164A</td>
<td>No</td>
<td>ATMEGA164A</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164P/PA</td>
<td>On-board</td>
<td>ATMEGA164P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168A</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168P/PA</td>
<td>On-board</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168PB</td>
<td>No</td>
<td>ATMEGA168PB</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>256KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega2561</td>
<td>No</td>
<td>ATMEGA2561</td>
<td>16MHz</td>
<td>256KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega32</td>
<td>No</td>
<td>ATMEGA32</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324A</td>
<td>On-board</td>
<td>ATMEGA324A</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324P</td>
<td>On-board</td>
<td>ATMEGA324P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PA</td>
<td>On-board</td>
<td>ATMEGA324PA</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PB</td>
<td>No</td>
<td>ATMEGA324PB</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328</td>
<td>On-board</td>
<td>ATMEGA328</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328P/PA</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328PB</td>
<td>No</td>
<td>ATMEGA328PB</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48/A</td>
<td>No</td>
<td>ATMEGA48</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48P/PA</td>
<td>On-board</td>
<td>ATMEGA48P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48PB</td>
<td>No</td>
<td>ATMEGA48PB</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega64/A</td>
<td>No</td>
<td>ATMEGA64</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega644/A</td>
<td>No</td>
<td>ATMEGA644A</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega644P/PA</td>
<td>On-board</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega8/A</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega8515</td>
<td>No</td>
<td>ATMEGA8515</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega8535</td>
<td>No</td>
<td>ATMEGA8535</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega88/A</td>
<td>On-board</td>
<td>ATMEGA88</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega88P/PA</td>
<td>On-board</td>
<td>ATMEGA88P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega88PB</td>
<td>No</td>
<td>ATMEGA88PB</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny13</td>
<td>On-board</td>
<td>ATINY13</td>
<td>9MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Atmel AT90PWM216</td>
<td>No</td>
<td>AT90PWM216</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Atmel AT90PWM316</td>
<td>No</td>
<td>AT90PWM316</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

**Microduino**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core (Atmega16PA@16M,5V)</td>
<td>On-board</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega16PA@8M,3.3V)</td>
<td>On-board</td>
<td>ATMEGA168P</td>
<td>8MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega32PA@16M,5V)</td>
<td>On-board</td>
<td>ATMEGA32P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega32PA@8M,3.3V)</td>
<td>On-board</td>
<td>ATMEGA32P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core USB (Atmega32U4@16M,5V)</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega128PA@16M,5V)</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega128PA@8M,3.3V)</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega64PA@16M,5V)</td>
<td>On-board</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega64PA@8M,3.3V)</td>
<td>On-board</td>
<td>ATMEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>
### OpenEnergyMonitor

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenEnergyMonitor emonPi</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### PanStamp

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanStamp AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Pinoccio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pinoccio Scout</td>
<td>No</td>
<td>ATMEGA256RFR2</td>
<td>16MHz</td>
<td>248KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Pololu Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pololu A-Star 32U4</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### Prusa 3D

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Prusa i3 MK3 Multi Material 2.0 Upgrade</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Prusa RAMBo</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### Punch Through

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightBlue Bean</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Quirkbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quirkbot</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

#### 1.10. Development Platforms
### RedBearLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBearLab Blend</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/16MHz (over-clock)</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/8MHz</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### RepRap

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepRap RAMBo</td>
<td>On-board</td>
<td>ATMega2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### SODAQ

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SODAQ GaLoRa</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Mbili</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Moja</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SODAQ Ndogo</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Tatu</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Sanguino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanguino ATmega1284p (16MHz)</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>On-board</td>
<td>ATMega644</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>On-board</td>
<td>ATMega644</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (16 MHz)</td>
<td>On-board</td>
<td>ATMega644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>On-board</td>
<td>ATMega644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>
### SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeeduino</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun ATmega128RFA1 Dev Board</td>
<td>On-board</td>
<td>ATMEGA128RFA1</td>
<td>16MHz</td>
<td>16KB</td>
<td>124KB</td>
</tr>
<tr>
<td>SparkFun Digital Sandbox</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Fio V3 3.3V/8MHz</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Makey Makey</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3.3V/8MHz</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 5V/16MHz</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro Mini 3.3V</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 3.3V/8MHz</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 5V/16MHz</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Qduino Mini</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### SpellFoundry

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpellFoundry Sleepy Pi 2</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### The Things Network

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Things Uno</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### Till Harbaum

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitDuino</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### TinyCircuits

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyCircuits TinyDuino Processor Board</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

1.10. Development Platforms
Wicked Device

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wicked Device WildFire V2</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>120.00KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V3</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Wisen

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk2 Whisper Node</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

makerlab.mx

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altair</td>
<td>No</td>
<td>ATMEGA256RFR2</td>
<td>16MHz</td>
<td>248KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

nicai-systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>nicai-systems BOB3 coding bot</td>
<td>On-board</td>
<td>ATMEGA88</td>
<td>8MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBO 2 robot</td>
<td>On-board</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot with Tuning Kit</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot with Tuning Kit</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

ubIQio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubIQio Ardhat</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Atmel megaAVR

Configuration `platform = atmelmegaavr`
8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

For more detailed information please visit vendor site.

Contents

- Configuration
- Examples
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Configuration

- Upload using Programmer
  - Upload using pyupdi
- Fuses programming
  - Custom fuses
  - Overriding default fuses command
- Bootloader programming
  - Custom bootloader
  - Overriding default bootloader command

Upload using Programmer

In the case of external programmers, it’s easy to brick a board simply by specifying incorrect upload flags. It’s highly recommended to use the upload_command option that gives the full control over flags used for uploading. Please read Upload using Programmer for more information.

Upload using pyupdi

pyupdi is a Python-based tool for programming tinyAVR and megaAVR devices with UPDI interface via a standard serial port. It can be installed directly in the PlatformIO virtual environment using the following command:

```
pip install https://github.com/mraardvark/pyupdi/archive/master.zip
```

Once pyupdi is installed it can be used as the uploader via a custom upload_command option, for example:
Warning: Device names used in `pyupdi` differ from MCU names used in the `atmelmegaavr` platform. Run `pyupdi --help` to see the list of supported devices.

More information and a typical circuit diagram can be found in the official `pyupdi` repository.

**Fuses programming**

PlatformIO has a built-in target called `fuses` for setting fuse bits. The default fuse bits are predefined in the board manifest file in the `fuses` section. For example, `fuses` section for Arduino Nano Every board. To set fuse bits you need to use target `fuses` with `pio run --target` command.

**Custom fuses**

Custom fuse values and upload flags (based on upload protocol) should be specified in the `platformio.ini` (Project Configuration File). An example of setting custom fuses for ATmega3209 board:

```
[env:custom_fuses]
platform = atmelmegaavr
framework = arduino
board = ATmega3209
upload_protocol = xplainedmini_updi
board_fuses.bootend = 0xAA
board_fuses.syscfg0 = 0xBB
board_fuses.osccfg = 0xCC
```

**Overriding default fuses command**

For more detailed information read `Overriding default fuses command`.

**Bootloader programming**

PlatformIO has a built-in target called `bootloader` for flashing bootloaders. The default bootloader image and corresponding fuse bits are predefined in the board manifest file in the `bootloader` section, for example, Arduino Uno WiFi Rev2. To upload a bootloader image you need to use target `bootloader` with `pio run --target` command.
Custom bootloader

Custom bootloader and accompanying fuses should be specified in “platformio.ini” (Project Configuration File). An example of setting custom bootloader for ATmega4808 board:

```
[env:ATmega4808]
platform = atmelmegaavr
framework = arduino
board = ATmega4808

board_bootloader.file = /path/to/custom/bootloader.hex
board_bootloader.bootend = 0xFF
board_bootloader.syscfg0 = 0xDE
board_bootloader.osccfg = 0xFD
board_bootloader.lock_bits = 0x0F
```

Overriding default bootloader command

For more detailed information read Overriding default bootloader command.

Examples

Examples are listed from Atmel megaAVR development platform repository:

- arduino-blink
- native-blink
- arduino-internal-libs
- arduino-external-libs

Stable and upstream versions

You can switch between stable releases of Atmel megaAVR development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

Stable

```
; Latest stable version
[env:latest_stable]
platform = atmelmegaavr
board = ...

; Custom stable version
[env:custom_stable]
platform = atmelmegaavr@x.y.z
board = ...
```
Upstream

```python
[env:upstream_develop]
platform = https://github.com/platformio/platform-atmelmegaavr.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduino-megaavr</td>
<td>Arduino Wiring-based Framework for Microchip megaAVR microcontrollers</td>
</tr>
<tr>
<td>framework-arduino-megaavr-megacorex</td>
<td>Arduino Wiring-based Framework for Microchip megaAVR microcontrollers (MegaCoreX)</td>
</tr>
<tr>
<td>framework-arduino-megaavr-megatinycore</td>
<td>Arduino Wiring-based Framework for Microchip tinyAVR 0-series and 1-series chips (megaTinyCore)</td>
</tr>
<tr>
<td>tool-avrdude-megaavr</td>
<td>AVRDUDE is a utility to download/upload/manipulate the ROM and EEPROM contents of megaAVR microcontrollers</td>
</tr>
<tr>
<td>toolchain-atmelavr</td>
<td>GCC Toolchain for Microchip AVR microcontrollers</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
- Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed `board` information please scroll the tables below by horizontally.
### Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Nano Every</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>47.50KB</td>
<td>6KB</td>
</tr>
<tr>
<td>Arduino Uno WiFi Rev2</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>47.50KB</td>
<td>6KB</td>
</tr>
</tbody>
</table>

### Microchip

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega1608</td>
<td>No</td>
<td>ATMEGA1608</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega1609</td>
<td>No</td>
<td>ATMEGA1609</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega3208</td>
<td>No</td>
<td>ATMEGA3208</td>
<td>16MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega3209</td>
<td>No</td>
<td>ATMEGA3209</td>
<td>16MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega4808</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ATmega4809</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ATmega808</td>
<td>No</td>
<td>ATMEGA808</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega809</td>
<td>No</td>
<td>ATMEGA809</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny1604</td>
<td>No</td>
<td>ATTINY1604</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny1606</td>
<td>No</td>
<td>ATTINY1606</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny1607</td>
<td>No</td>
<td>ATTINY1607</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny1614</td>
<td>No</td>
<td>ATTINY1614</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny1616</td>
<td>No</td>
<td>ATTINY1616</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny1617</td>
<td>No</td>
<td>ATTINY1617</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny202</td>
<td>No</td>
<td>ATTINY202</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny204</td>
<td>No</td>
<td>ATTINY204</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny212</td>
<td>No</td>
<td>ATTINY212</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny214</td>
<td>No</td>
<td>ATTINY214</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny3216</td>
<td>No</td>
<td>ATTINY3216</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny3217</td>
<td>No</td>
<td>ATTINY3217</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny402</td>
<td>No</td>
<td>ATTINY402</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny404</td>
<td>No</td>
<td>ATTINY404</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny406</td>
<td>No</td>
<td>ATTINY406</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny412</td>
<td>No</td>
<td>ATTINY412</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny414</td>
<td>No</td>
<td>ATTINY414</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny416</td>
<td>No</td>
<td>ATTINY416</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny417</td>
<td>No</td>
<td>ATTINY417</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny804</td>
<td>No</td>
<td>ATTINY804</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny806</td>
<td>No</td>
<td>ATTINY806</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny807</td>
<td>No</td>
<td>ATTINY807</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny814</td>
<td>No</td>
<td>ATTINY814</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny816</td>
<td>No</td>
<td>ATTINY816</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny817</td>
<td>No</td>
<td>ATTINY817</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>AVR-IoT WG Development Board</td>
<td>No</td>
<td>ATMEGA4808</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>Curiosity Nano ATmega4809</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>Xplained Pro ATmega4809</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
</tbody>
</table>

### Atmel SAM

Configuration `platform = atmelsam`

1.10. Development Platforms
Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

For more detailed information please visit vendor site.

Contents

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Examples

Examples are listed from Atmel SAM development platform repository:

- mbed-blink
- mbed-serial
- zephyr-blink
- zephyr-subsys-logger
- mbed-events
- arduino-blink
- simba-blink
- arduino-internal-libs
- zephyr-drivers-lcd-hd44780
- arduino-external-libs
- mbed-dsp
- arduino-web-thing-led
- arduino-brika-internal-libs

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools
  - External Debug Tools
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMC21-XPRO</td>
<td>SARM21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>SAML21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>SAML21J18B</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

External Debug Tools

Boards listed below are compatible with Debugging but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit BLM Badge</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Crickit M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 CAN</td>
<td>SAME51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>SAMD51P20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M4</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit MONSTER M4SK</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Matrix Portal M4</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M0 Express</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M4 AirLift Lite</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4 Titano</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Trellis M4</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pyIRkey</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pyBadge AirLift M4</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>1008KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit pyBadge M4 Express</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1310</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - Samd21</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Briki MBC-WB - Samd21</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Minitronics v2.0</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ Autonomo</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ExpLoRer</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Seeeduino Femto M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Terminal</td>
<td>SAMD51P19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun 9DoF Razor IMU M0</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun Qwiic Micro</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Continued on next page
### Stable and upstream versions

You can switch between stable releases of Atmel SAM development platform and the latest upstream version using `platform` option in “`platformio.ini` (Project Configuration File)” as described below.

#### Stable

```ini
; Latest stable version
[env:latest_stable]
platform = atmelsam
board = ...

; Custom stable version
[env:custom_stable]
platform = atmelsam@x.y.z
board = ...
```

#### Upstream

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-atmelsam.git
board = ...
```

### Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduino-mbcwb</td>
<td>Fork of Arduino Framework for briki MBC-WB boards</td>
</tr>
<tr>
<td>framework-arduino-sam</td>
<td>The official Arduino Wiring-based Framework for ATSAM3 microcontrollers</td>
</tr>
<tr>
<td>framework-arduino-samd</td>
<td>The official Arduino Wiring-based Framework for Microchip SAM D microcontrollers</td>
</tr>
<tr>
<td>framework-arduino-samd-adafruit</td>
<td>Arduino Wiring-based Framework for Microchip SAM D microcontrollers (Adafruit SAMD)</td>
</tr>
<tr>
<td>framework-arduino-samd-moteino</td>
<td>Arduino Wiring-based Framework for Microchip SAM D microcontrollers (Moteino SAMD)</td>
</tr>
<tr>
<td>framework-arduino-samd-reprap</td>
<td>Arduino Wiring-based Framework for Microchip SAM D microcontrollers (RepRap SAMD)</td>
</tr>
<tr>
<td>framework-arduino-samd-seeed</td>
<td>Arduino Wiring-based Framework for Microchip SAM D microcontrollers (Seeed SAMD)</td>
</tr>
<tr>
<td>framework-arduino-samd-sodaq</td>
<td>Arduino Wiring-based Framework for Microchip SAM D microcontrollers (SODAQ SAMD)</td>
</tr>
<tr>
<td>framework-arduino-samd-sparkfun</td>
<td>Arduino Wiring-based Framework for Microchip SAM D microcontrollers (SparkFun SAMD)</td>
</tr>
<tr>
<td>framework-arduino-samd-tuino0</td>
<td>Arduino Wiring-based Framework for Microchip SAM D microcontrollers (Tuino0 SAMD)</td>
</tr>
<tr>
<td>framework-cmsis</td>
<td>Vendor-independent hardware abstraction layer for the Cortex-M processor series</td>
</tr>
<tr>
<td>framework-cmsis-atmel</td>
<td>Atmel Smart ARM devices CMSIS module</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba is an Embedded Programming Platform. It aims to make embedded programming easy</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-atmel</td>
<td>Atmel SAM HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinybor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARM</td>
</tr>
<tr>
<td>tool-avrdude</td>
<td>AVRDUDE is a utility to download/upload/manipulate the ROM and EEPROM contents of a device</td>
</tr>
<tr>
<td>tool-bossac</td>
<td>Basic Open Source SAM-BA Application (BOSSA)</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-mbctool</td>
<td>MBC-WB Uploader Application</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing</td>
</tr>
<tr>
<td>toolchain-gccarmnoneneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

**Warning: Linux Users:**

- Install "udev" rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

**Windows Users:**

Please check that you have a correctly installed USB driver from board manufacturer.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit BLM Badge</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 CAN</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>External</td>
<td>SAMD51P20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M4</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit MONSTER M4SK</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Matrix Portal M4</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M0 Expresss</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4 Titano</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Trellis M4</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pIRkey</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pyBadge AirLift M4</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1008KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit pyBadge M4 Express</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>
## Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1310</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR ZERO</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Atmel

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMC21-XPRO</td>
<td>On-board</td>
<td>SAMC21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Digistump

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digistump DigiX</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

## Gimasi

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuino 096</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
### LowPowerLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowPowerLab CurrentRanger</td>
<td>No</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### ReprapWorld

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minitronics v2.0</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### SODAQ

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SODAQ Autonomo</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ExpLoRer</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### SainSmart

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

### Seeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeeduino Femto M0</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Terminal</td>
<td>External</td>
<td>SAMD51P19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun 9DoF Razor IMU M0</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun Qwiic Micro</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD51 Thing Plus</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

meteca

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Briki ABC (MBC-WB) - Samd21</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Briki MBC-WB - Samd21</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

CHIPS Alliance

**Configuration**  
`platform = chipsalliance`

The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs. For more detailed information please visit vendor site.

Contents

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Examples

Examples are listed from CHIPS Alliance development platform repository:

- native-blink_asm
- native-bare_c
- rtosal-freertos
- native asm
- zephyr-hello-world
- psp-hello-world
Debugging

"1-click" solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - **On-Board Debug Tools**

**Tools & Debug Probes**

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

**On-Board Debug Tools**

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
</tbody>
</table>

**Stable and upstream versions**

You can switch between stable releases of CHIPS Alliance development platform and the latest upstream version using `platform` option in “platformio.ini” (*Project Configuration File*) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = chipsalliance
board = ...

; Custom stable version
[env:custom_stable]
platform = chipsalliance@x.y.z
board = ...
```
Upstream

[env:upstream_develop]
platform = https://github.com/platformio/platform-chipsalliance.git
board = ...

1.10. Development Platforms
## Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-wd-riscv-sdk</td>
<td>The WD Firmware package contains Firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features.</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures.</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>Zephyr module for CANopen node</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-swervolf</td>
<td>SweRVolf is a FuseSoC-based SoC for the SweRV RISC-V core.</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-lorawan-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-nipisys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinycbor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of standard cryptography primitives for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARMv8-M and Zephyr framework</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-openocd-riscv-chipsalliance</td>
<td>Fork of Open On-Chip Debugger that has RISC-V support and enabled VPI JTAG</td>
</tr>
<tr>
<td>tool-verilator-swervolf</td>
<td>Verilator is an open-source SystemVerilog simulator and lint system</td>
</tr>
<tr>
<td>tool-whisper</td>
<td>Whisper is a RISCV instruction set simulator (ISS) developed for the verification of the SweRV micro-controller. It allows the user to run RISC-V code without RISC-V hardware</td>
</tr>
<tr>
<td>toolchain-riscv</td>
<td>GNU toolchain for RISC-V, including GCC</td>
</tr>
</tbody>
</table>
**Warning:** Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

**Windows Users:**
Please check that you have a correctly installed USB driver from board manufacturer.

---

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FreeRTOS</strong></td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td><strong>WD-Firmware</strong></td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td><strong>Zephyr RTOS</strong></td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Boards

Note:
- You can list pre-configured boards by `pio boards` command or [PlatformIO Boards Explorer](#).
- For more detailed board information please scroll the tables below by horizontally.

---

#### Digilent

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>RVfpga: Digilent Nexys A7</em></td>
<td>On-board</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
</tbody>
</table>

#### Espressif 32

**Configuration** `platform = espressif32`

Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

For more detailed information please visit [vendor site](#).

---

### Contents

- Tutorials
- Configuration

---

1.10. Development Platforms 377
• Examples
• Debugging
• Stable and upstream versions
• Packages
• Frameworks
• Boards

Tutorials

• Get started with Arduino and ESP32-DevKitC: debugging and unit testing
• Get started with ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis
• Video: Free Inline Debugging for ESP32 and Arduino Sketches

Configuration

• CPU Frequency
• FLASH Frequency
• FLASH Mode
• External RAM (PSRAM)
• Debug Level
• Upload Speed
• Erase Flash
• Partition Tables
• Embedding Binary Data
• uploading files to file system SPIFFS
• Over-the-Air (OTA) update
  – Using JFrog Bintray (free and secure Cloud solution)
  – Using built-in Local solution
    * Authentication and upload options
• Using Arduino Framework with Staging version
• Arduino Core Wiki

CPU Frequency

See board_build.f_cpu option from “platformio.ini” (Project Configuration File)
FLASH Frequency

Please use `board_build.f_flash` option from "platformio.ini" (Project Configuration File) to change a value. Possible values:

- 40000000L (default)
- 80000000L

FLASH Mode

Flash chip interface mode. This parameter is stored in the binary image header, along with the flash size and flash frequency. The ROM bootloader in the ESP chip uses the value of these parameters in order to know how to talk to the flash chip.

Please use `board_build.flash_mode` option from "platformio.ini" (Project Configuration File) to change a value. Possible values:

- qio
- qout
- dio
- dout

External RAM (PSRAM)

You can enable external RAM using the next extra `build_flags` in "platformio.ini" (Project Configuration File) depending on a framework type.

Framework Arduino:

```ini
[env:myenv]
platform = espressif32
framework = arduino
board = ...
build_flags =
   -DBOARD_HAS_PSRAM
   -mfix-esp32-psram-cache-issue
```

Framework Espressif IoT Development Framework:
More details are located in the official ESP-IDF documentation - Support for external RAM.

**Debug Level**

Please use one of the next `build_flags` to change debug level. A `build_flags` option could be used only the one time per build environment. If you need to specify more flags, please separate them with a new line or space.

Actual information is available in Arduino for ESP32 Board Manifest. Please scroll to `esp32.menu.DebugLevel` section.

```ini
[env:myenv]
platform = espressif32
framework = espidf
board = ...
build_flags =
    -DCONFIG_SPIRAM_CACHE_WORKAROUND
```

```ini
[env:myenv]
platform = ...
board = ...
framework = arduino

;;;;; Possible options ;;;;;;

// None
build_flags = -DCORE_DEBUG_LEVEL=0

// Error
build_flags = -DCORE_DEBUG_LEVEL=1

// Warn
build_flags = -DCORE_DEBUG_LEVEL=2

// Info
build_flags = -DCORE_DEBUG_LEVEL=3

// Debug
build_flags = -DCORE_DEBUG_LEVEL=4

// Verbose
build_flags = -DCORE_DEBUG_LEVEL=5
```

**Upload Speed**

You can set custom upload speed using `upload_speed` option from “platformio.ini” (Project Configuration File)

```ini
[env:myenv]
upload_speed = 9600
```

**Erase Flash**

Please `pio run --target` the next command to erase the entire flash chip (all data replaced with 0xFF bytes):
> pio run --target erase
# or short version
> pio run -t erase

Partition Tables

You can create a custom partitions table (CSV) following ESP32 Partition Tables documentation. PlatformIO uses default partition tables depending on a framework type:
- `default.csv` for Arduino (show pre-configured partition tables)
- `partitions_singleapp.csv` for Espressif IoT Development Framework (show pre-configured partition tables)

To override default table please use `board_build.partitions` option in "platformio.ini" (Project Configuration File).

**Warning:** SPIFFS partition MUST have configured “Type” as “data” and “SubType” as “spiffs”. For example, `spiffs, data, spiffs, 0x291000, 1M`,

Examples:

```plaintext
; 1) A "partitions_custom.csv" in the root of project directory
[env:custom_table]
board_build.partitions = partitions_custom.csv

; 2) Switch between built-in tables
; https://github.com/espressif/arduino-esp32/tree/master/tools/partitions
; https://github.com/espressif/esp-idf/tree/master/components/partition_table
[env:custom_builtin_table]
board_build.partitions = no_ota.csv
```

Embedding Binary Data

Sometimes you have a file with some binary or text data that you’d like to make available to your program - but you don’t want to reformat the file as C source.

There are two options `board_build.embed_txtfiles` and `board_build.embed_files` which can be used for embedding data. The only difference is that files specified in `board_build.embed_txtfiles` option are null-terminated in the final binary.

```plaintext
[env:myenv]
platform = espressif32
board = ...
board_build.embed_txtfiles =
  src/private.pem.key
  src/certificate.pem.crt
  src/aws-root-ca.pem
```

The file contents will be added to the .rodata section in flash, and are available via symbol names as follows:
extern const uint8_t aws_root_ca_pem_start[] asm("_binary_src_aws_root_ca_pem_start");
extern const uint8_t aws_root_ca_pem_end[] asm("_binary_src_aws_root_ca_pem_end");
extern const uint8_t certificate_pem_crt_start[] asm("_binary_src_certificate_pem_crt_start");
extern const uint8_t certificate_pem_crt_end[] asm("_binary_src_certificate_pem_crt_end");
extern const uint8_t private_pem_key_start[] asm("_binary_src_private_pem_key_start");
extern const uint8_t private_pem_key_end[] asm("_binary_src_private_pem_key_end");

The names are generated from the full name of the file. Characters /, ., etc. are replaced with underscores. The _binary + _nested_folder prefix in the symbol name is added by “objcopy” and is the same for both text and binary files.

Note: With the ESP-IDF framework symbol names should not contain path to the files, for example _binary_private_pem_key_start instead of _binary_src_private_pem_key_start.

See full example with embedding Amazon AWS certificates:

- https://github.com/platformio/platform-espressif32/tree/develop/examples/espidf-aws-iot

Uploading files to file system SPIFFS

1. Create new project using PlatformIO IDE or initialize project using PlatformIO Core (CLI) and pio project init (if you have not initialized it yet)
2. Create data folder (it should be on the same level as src folder) and put files here. Also, you can specify own location for data_dir
3. Run “Upload File System image” task in PlatformIO IDE or use PlatformIO Core (CLI) and pio run --target command with uploadfs target.

To upload SPIFFS image using OTA update please specify upload_port / --upload-port as IP address or mDNS host name (ending with the *.local).

Examples:

- SPIFFS for Arduino
- SPIFFS for ESP-IDF

Over-the-Air (OTA) update

Using JFrog Bintray (free and secure Cloud solution)

- Video and presentation - swampUP: Over-The-Air (OTA) firmware upgrades for Internet of Things devices with PlatformIO and JFrog Bintray
- Demo source code: https://github.com/platformio/bintray-secure-ota

Using built-in Local solution

Demo code for:

- Arduino
• ESP-IDF

There are 2 options:

• Directly specify `pio run --upload-port` in command line

```
pio run --target upload --upload-port IP_ADDRESS_HERE or mDNS_NAME.local
```

• Specify `upload_port` option in “`platformio.ini`” (Project Configuration File)

You also need to set `upload_protocol` to `espota`.

```
[env:myenv]
upload_protocol = espota
upload_port = IP_ADDRESS_HERE or mDNS_NAME.local
```

For example,

• `pio run -t upload --upload-port 192.168.0.255`
• `pio run -t upload --upload-port myesp8266.local`

Authentication and upload options

You can pass additional options/flags to OTA uploader using `upload_flags` option in “`platformio.ini`” (Project Configuration File)

```
[env:myenv]
upload_protocol = espota
; each flag in a new line
upload_flags =
    --port=3232
```

Available flags

- `--port=ESP_PORT` ESP32 OTA Port. **Default 8266**
- `--auth=AUTH` Set authentication password
- `--spiffs` Use this option to transmit a SPIFFS image and do not flash the module

For the full list with available options please run

```
~/.platformio/packages/framework-arduinoespressif32/tools/espota.py --help
```

Usage: espota.py [options]

Transmit image over the air to the esp32 module with OTA support.

Options:

- `-h, --help` show this help message and exit

Destination:

- `-i ESP_IP, --ip=ESP_IP` ESP32 IP Address.
- `-I HOST_IP, --host_ip=HOST_IP` Host IP Address.
- `-p ESP_PORT, --port=ESP_PORT` ESP32 ota Port. Default **3232**
- `-P HOST_PORT, --host_port=HOST_PORT`

(continues on next page)
Host server ota Port. Default random 10000-60000

Authentication:
-a AUTH, --auth=AUTH
Set authentication password.

Image:
-f FILE, --file=FILE
Image file.
-s, --spiffs Use this option to transmit a SPIFFS image and do not flash the module.

Output:
-d, --debug Show debug output. And override loglevel with debug.
-r, --progress Show progress output. Does not work for ArduinoIDE
-t TIMEOUT, --timeout=TIMEOUT Timeout to wait for the ESP32 to accept invitation

Warning: For windows users. To manage OTA check the ESP wifi network profile isn’t checked on public be sure it’s on private mode’’

Using Arduino Framework with Staging version

PlatformIO will install the latest Arduino Core for ESP32 from https://github.com/espressif/arduino-esp32. The Git should be installed in a system. To update Arduino Core to the latest revision, please open PlatformIO IDE and navigate to PlatformIO Home > Platforms > Updates.

1. Please install PlatformIO IDE
2. Initialize a new project, open “platformio.ini” (Project Configuration File) and specify the link to the framework repository in platform_packages section. For example,

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
platform_packages =
    platformio/framework-arduinospressif32 @ https://github.com/espressif/arduino-esp32.git
```
3. Try to build the project
4. If you see build errors, then try to build this project using the same stage with Arduino IDE
5. If it works with Arduino IDE but doesn’t work with PlatformIO, then please file a new issue with attached information:
   - test project/files
   - detailed log of build process from Arduino IDE (please copy it from console to https://hastebin.com)
   - detailed log of build process from PlatformIO Build System (please copy it from console to https://hastebin.com)
Arduino Core Wiki


Examples

Examples are listed from Espressif 32 development platform repository:

- espidf-storage-sdcard
- espidf-blink
- espidf-coap-server
- espidf-exceptions
- arduino-blink
- simba-blink
- espidf-ble-eddystone
- espidf-ulp-adc
- espidf-ulp-pulse
- espidf-arduino-wifiscan
- espidf-http-request
- espidf-arduino-blink
- pumbaa-blink
- espidf-hello-world
- espidf-aws-iot
- arduino-briki-internal-libs
- espidf-peripherals-uart
- arduino-wifiscan

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Pinout Diagram
- Tools & Debug Probes
  - On-Board Debug Tools
  - External Debug Tools
Pinout Diagram

JTAG Wiring Connections

<table>
<thead>
<tr>
<th>Board Pin</th>
<th>JTAG Tool Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO13</td>
<td>TCK</td>
</tr>
<tr>
<td>IO12</td>
<td>TDI</td>
</tr>
<tr>
<td>IO15</td>
<td>TDO</td>
</tr>
<tr>
<td>IO14</td>
<td>TMS</td>
</tr>
<tr>
<td>EN</td>
<td>RST</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
External Debug Tools

Boards listed below are compatible with **Debugging** but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-duino-32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32n IoT Uno</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IoTaP Magnolia</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32 DevKit-LiPo</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
Table 7 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Espressif 32 development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

Stable

```ini
; Latest stable version
[env:latest_stable]
platform = espressif32
board = ...

; Custom stable version
[env:custom_stable]
platform = espressif32@x.y.z
board = ...
```

Upstream

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-espressif32.git
board = ...
```
## Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduino-mbcwb</td>
<td>Fork of Arduino Framework for briki MBC-WB boards</td>
</tr>
<tr>
<td>framework-arduinoespressif32</td>
<td>Arduino Wiring-based Framework for Espressif ESP32 microcontrollers</td>
</tr>
<tr>
<td>framework-espidf</td>
<td>Espressif IoT Development Framework. Official development framework for ESP32 chip</td>
</tr>
<tr>
<td>framework-pumbaa</td>
<td>Pumbaa Framework - a port of MicroPython, designed for embedded devices with limited amount of RAM and code memory</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba is an Embedded Programming Platform. It aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-esptoolpy</td>
<td>Espressif ESP8266 and ESP32 serial bootloader utility</td>
</tr>
<tr>
<td>tool-idf</td>
<td>idf is a top-level config/build command line tool for ESP-IDF</td>
</tr>
<tr>
<td>tool-mbctool</td>
<td>MBC-WB Uploader Application</td>
</tr>
<tr>
<td>tool-mconf</td>
<td>Fork of kconfig-frontends project with some modifications for use with ESP-IDF</td>
</tr>
<tr>
<td>tool-mksipi</td>
<td>Tool to build and unpack SPIFFS images</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-openocd-esp32</td>
<td>Open On-Chip Debugger for Espressif ESP32</td>
</tr>
<tr>
<td>toolchain-esp32s2ulp</td>
<td>Binutils fork with support for the ESP32-S2 ULP co-processor</td>
</tr>
<tr>
<td>toolchain-esp32ulp</td>
<td>Binutils fork with support for the Espressif ESP32 ULP co-processor</td>
</tr>
<tr>
<td>toolchain-xtensa32</td>
<td>GCC Toolchain for Xtensa32 processor</td>
</tr>
<tr>
<td>toolchain-xtensa32s2</td>
<td>GCC Toolchain for Xtensa32-S2 processor</td>
</tr>
</tbody>
</table>

**Warning: Linux Users:**

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

**Windows Users:**

Please check that you have a correctly installed USB driver from board manufacturer

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td><strong>Pumbaa</strong></td>
<td>Pumbaa is Python on top of Simba. The implementation is a port of MicroPython, designed for embedded devices with limited amount of RAM and code memory</td>
</tr>
<tr>
<td><strong>Simba</strong></td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>
Board

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

AI Thinker

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

AZ-Delivery

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
</tbody>
</table>

Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Aiyarafun

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node32s</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

April Brother

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>April Brother ESPea32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

BPI Tech

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPI-Bit</td>
<td>No</td>
<td>ESP32</td>
<td>160MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### DFRobot

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireBeetle-ESP32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### DOIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### DSTIKE

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-duino-32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Dongsen Technology

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### DycodeX

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPectro32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### ESP32vn

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32vn IoT Uno</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Electronic SweetPeas

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronic SweetPeas ESP320</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32 Pico Kit</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Fred

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frog Board ESP32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Hardkernel

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODROID-GO</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Heltec Automation

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec WiFi Kit 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Hornbill

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## IntoRobot

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntoRobot Fig</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## IoTaaP

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoTaaP Magnolia</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## M5Stack

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M5Stack Core ESP32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>M5Stack FIRE</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>6.25MB</td>
</tr>
<tr>
<td>M5Stack GREY ESP32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>M5Stick-C</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## MH-ET Live

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Magicblocks.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagicBit</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## MakerAsia

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakerAsia Nano32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core ESP32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU-32S</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Noduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noduino Quantum</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PRO</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PoE</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PoE-ISO</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Oroca

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OROCA EduBot</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Onehorse

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onehorse ESP32 Dev Module</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Pycom Ltd.

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pycom GPy</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>1.25MB</td>
<td></td>
</tr>
</tbody>
</table>

### Qmobot LLP

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qchip</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### RoboticsBrno

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALKS ESP32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SG-O

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG-O AirMon</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### Silicognition

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silicognition wESP32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SparkFun Electronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### TTGO

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T-Watch</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### ThaiEasyElec

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### TinyPICO

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyPICO</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Turta

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turta IoT Node</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### Unknown

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32 FM DevKit</td>
<td></td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### VintLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td></td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### WEMOS

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS LOLIN D32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Widora

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Widora AIR</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### XinaBox

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinaBox CW02</td>
<td></td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### YeaCreate

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>YeaCreate NSCREEN-32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### meteca

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
oddWires

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-blox NINA-W10 series</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>2MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Espressif 8266

Configuration `platform = espressif8266`

Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

For more detailed information please visit vendor site.

Contents

- Configuration
- Examples
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Configuration

- CPU Frequency
- FLASH Frequency
- FLASH Mode
- Reset Method
- Flash Size
- Upload Speed
- lwIP Variant
- SDK Version
- SSL Support
• Serial Debug
• Debug Level
• VTables
• Exceptions
• Using Filesystem
  – Selecting appropriate Filesystem
  – Uploading files to Filesystem
  – Overridding Filesystem image name
• Over-the-Air (OTA) update
  – Authentication and upload options
• Demo
• Using Arduino Framework with Staging version

CPU Frequency

See board_build.f_cpu option from “platformio.ini” (Project Configuration File)

```
[env:myenv]
; set frequency to 160MHz
board_build.f_cpu = 160000000L
```

FLASH Frequency

Please use board_build.f_flash option from “platformio.ini” (Project Configuration File) to change a value. Possible values:

- 20000000L
- 26000000L
- 40000000L (default)
- 80000000L

```
[env:myenv]
; set frequency to 80MHz
board_build.f_flash = 80000000L
```

FLASH Mode

Flash chip interface mode. This parameter is stored in the binary image header, along with the flash size and flash frequency. The ROM bootloader in the ESP chip uses the value of these parameters in order to know how to talk to the flash chip.

Please use board_build.flash_mode option from “platformio.ini” (Project Configuration File) to change a value. Possible values:
• qio
• qout
• dio
• dout

```
[env:myenv]
board_build.flash_mode = qio
```

**Reset Method**

You can set custom reset method using `upload_resetmethod` option from “`platformio.ini`” (Project Configuration File).

The possible values are:

- **ck** - RTS controls RESET or CH_PD, DTR controls GPIO0
- **wifio** - TXD controls GPIO0 via PNP transistor and DTR controls RESET via a capacitor
- **nodemcu** - GPIO0 and RESET controlled using two NPN transistors as in NodeMCU devkit.

See default reset methods per board.

```
[env:myenv]
upload_resetmethod = ck
```

**Flash Size**

**Warning:** Please make sure to read ESP8266 Flash layout information first.

Available LD-scripts: https://github.com/esp8266/Arduino/tree/master/tools/sdk/ld

Please open `eagle.flash.***.ld` file to check how flash is split.

To override default LD script please use `board_build.ldscript` option from “`platformio.ini`” (Project Configuration File).

```
[env:myenv]
board_build.ldscript = eagle.flash.4m.ld
```

**Upload Speed**

You can set custom upload speed using `upload_speed` option from “`platformio.ini`” (Project Configuration File)

```
[env:myenv]
upload_speed = 9600
```

**lwIP Variant**

Available variants (macros):
• `-D PIO_FRAMEWORK_ARDUINO_LWIP2_LOW_MEMORY` v2 Lower Memory (default)
• `-D PIO_FRAMEWORK_ARDUINO_LWIP2_HIGHER_BANDWIDTH` v2 Higher Bandwidth
• `-D PIO_FRAMEWORK_ARDUINO_LWIP2_LOW_MEMORY_LOW_FLASH` v2 Lower Memory (no features)
• `-D PIO_FRAMEWORK_ARDUINO_LWIP2_HIGHER_BANDWIDTH_LOW_FLASH` v2 Higher Bandwidth (no features)
• `-D PIO_FRAMEWORK_ARDUINO_LWIP2_IPV6_LOW_MEMORY` v2 IPv6 Lower Memory
• `-D PIO_FRAMEWORK_ARDUINO_LWIP2_IPV6_HIGHER_BANDWIDTH` v2 IPv6 Higher Bandwidth
• `-D PIO_FRAMEWORK_ARDUINO_LWIP_HIGHER_BANDWIDTH` v1.4 Higher Bandwidth

You can change lwIP Variant by passing a custom macro using project `build_flags`.

For example, to switch to lwIP v1.4

```bash
[env:myenv]
...
build_flags = -D PIO_FRAMEWORK_ARDUINO_LWIP_HIGHER_BANDWIDTH
```

### SDK Version

Available versions (macros):

• `-D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK3` NonOS SDK-pre-3.0 as of Jun 26, 2018
• `-D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK221` NonOS SDK v2.2.1 (legacy) as of Jun 8, 2018
• `-D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK22x_190313` NonOS SDK v2.2.x branch as of Mar 13, 2019
• `-D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK22x_190703` NonOS SDK v2.2.x branch as of Jul 03, 2019 (default)
• `-D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK22x_191024` NonOS SDK v2.2.x branch as of Oct 24, 2019
• `-D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK22x_191105` NonOS SDK v2.2.x branch as of to Nov 05, 2019
• `-D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK22x_191122` NonOS SDK v2.2.x branch as of to Nov 22, 2019

You can change SDK version by passing a custom macro using project `build_flags`.

For example, to switch to SDK-pre-3.0:

```bash
[env:myenv]
...
build_flags = -D PIO_FRAMEWORK_ARDUINO_ESPRESSIF_SDK3
```

### SSL Support

By default, all SSL ciphers (most compatible) are supported.

You can control SSL support passing a custom macro using project `build_flags`.

For example, use basic SSL ciphers (lower ROM use):
Serial Debug

Please use the next `build_flags` to enable Serial debug:

```plaintext
[env:myenv]
...
build_flags = -D DEBUG_ESP_PORT=Serial
; or for Serial1
build_flags = -D DEBUG_ESP_PORT=Serial1
```

Debug Level

Please use one of the next `build_flags` to change debug level. A `build_flags` option could be used only the one time per build environment. If you need to specify more flags, please separate them with a new line or space.

Also, please note that you will need to extend `build_flags` with `Serial Debug` macro. For example, `build_flags = -D DEBUG_ESP_PORT=Serial -D DEBUG_ESP_SSL ...`

Actual information is available in Arduino for ESP8266 Board Manifest. Please scroll to `generic.menu.lvl` section.

```plaintext
[env:myenv]
platform = ...
board = ...
framework = arduino

;;;;; Possible options ;;;;;

; SSL
build_flags = -D DEBUG_ESP_SSL

; TLS_MEM
build_flags = -D DEBUG_ESP_TLS_MEM

; HTTP_CLIENT
build_flags = -D DEBUG_ESP_HTTP_CLIENT

; HTTP_SERVER
build_flags = -D DEBUG_ESP_HTTP_SERVER

; SSL+TLS_MEM
build_flags =
  -D DEBUG_ESP_SSL
  -D DEBUG_ESP_TLS_MEM

; SSL+HTTP_CLIENT
build_flags =
  -D DEBUG_ESP_SSL
  -D DEBUG_ESP_HTTP_CLIENT
```

(continues on next page)
; SSL+HTTP_SERVER
build_flags =
  -DDEBUG_ESP_SSL
  -DDEBUG_ESP_HTTP_SERVER

; TLS_MEM+HTTP_CLIENT
build_flags =
  -DDEBUG_ESP_TLS_MEM
  -DDEBUG_ESP_HTTP_CLIENT

; TLS_MEM+HTTP_SERVER
build_flags =
  -DDEBUG_ESP_TLS_MEM
  -DDEBUG_ESP_HTTP_SERVER

; HTTP_CLIENT+HTTP_SERVER
build_flags =
  -DDEBUG_ESP_HTTP_CLIENT
  -DDEBUG_ESP_HTTP_SERVER

; SSL+TLS_MEM+HTTP_CLIENT
build_flags =
  -DDEBUG_ESP_SSL
  -DDEBUG_ESP_TLS_MEM
  -DDEBUG_ESP_HTTP_CLIENT

; SSL+TLS_MEM+HTTP_SERVER
build_flags =
  -DDEBUG_ESP_SSL
  -DDEBUG_ESP_TLS_MEM
  -DDEBUG_ESP_HTTP_SERVER

; SSL+HTTP_CLIENT+HTTP_SERVER
build_flags =
  -DDEBUG_ESP_SSL
  -DDEBUG_ESP_HTTP_CLIENT
  -DDEBUG_ESP_HTTP_SERVER

; TLS_MEM+HTTP_CLIENT+HTTP_SERVER
build_flags =
  -DDEBUG_ESP_TLS_MEM
  -DDEBUG_ESP_HTTP_CLIENT
  -DDEBUG_ESP_HTTP_SERVER

; SSL+TLS_MEM+HTTP_CLIENT+HTTP_SERVER
build_flags =
  -DDEBUG_ESP_SSL
  -DDEBUG_ESP_TLS_MEM
  -DDEBUG_ESP_HTTP_CLIENT
  -DDEBUG_ESP_HTTP_SERVER

; CORE
build_flags = -DDEBUG_ESP_CORE

; WIFI
build_flags = -DDEBUG_ESP_WIFI
VTables

Please use one of the next `build_flags`:

```sh
[env:myenv]
...

; Flash (default)
build_flags = -DVTABLES_IN_FLASH

; Heap
build_flags = -DVTABLES_IN_DRAM

; IRAM
build_flags = -DVTABLES_IN_IRAM
```
Exceptions

Exceptions are disabled by default. To enable exceptions, use the following build_flags and build_unflags:

```plaintext
[env:myenv]
...
; Remove default exceptions disabled flag
build_unflags = -fno-exceptions
; Enable exceptions
build_flags = -fexceptions
```

Using Filesystem

Selecting appropriate Filesystem

There are two file systems for utilizing the on-board flash on the ESP8266: SPIFFS and LittleFS. They provide a compatible API but have incompatible on-flash implementations, so it is important to choose one or the per project as attempting to mount a SPIFFS volume under LittleFS may result in a format operation and definitely will not preserve any files, and vice-versa.

**Warning:** SPIFFS is currently deprecated and may be removed in future releases of the core. Please consider moving your code to LittleFS.

The SPIFFS file system is used by default in order to keep legacy project compatible. To choose LittleFS as the file system, it should be explicitly specified using board_build.filesystem option in “platformio.ini” (Project Configuration File), for example:

```plaintext
[env:myenv]
platform = espressif8266
framework = arduino
board = ...
board_build.filesystem = littlefs
```

More information about pros and cons of each file system can be found in the official documentation.

Uploading files to Filesystem

**Warning:** Please make sure to read ESP8266 Flash layout information first.

1. Create new project using PlatformIO IDE or initialize project using PlatformIO Core (CLI) and pio project init (if you have not initialized it yet)
2. Create data folder (it should be on the same level as src folder) and put files here. Also, you can specify own location for data_dir
3. Run “Upload File System image” task in PlatformIO IDE or use PlatformIO Core (CLI) and pio run --target command with uploadfs target.
To upload file system image using OTA update please specify `upload_port` as IP address or mDNS host name (ending with the `*.local`). For the details please follow to Over-the-Air (OTA) update.

By default, will be used default LD Script for the board where is specified file system offsets (start, end, page, block). You can override it using `Flash Size`.

Active discussion is located in issue #382.

**Overriding Filesystem image name**

By default, the image file name is set according to the used file system: `spiffs.bin` or `littlefs.bin`. You can change the file name using a PRE extra script, for example:

```plaintext
[env:d1]
platform = espressif8266
framework = arduino
board = d1
board_build.filesystem = littlefs
extra_scripts =
    pre:extra_script.py
```

Where a special variable `ESP8266_FS_IMAGE_NAME` can be overridden:

```python
import("env")
env.Replace(ESP8266_FS_IMAGE_NAME="custom_image_name")
```

**Over-the-Air (OTA) update**

Firstly, please read What is OTA? How to use it?

There are 2 options:

- Directly specify `pio run --upload-port` in command line

  ```bash
  pio run --target upload --upload-port IP_ADDRESS_HERE or mDNS_NAME.local
  ```

- Specify `upload_port` option in "`platformio.ini`" (Project Configuration File)

  You also need to set `upload_protocol` to `espota`.

  ```plaintext
  [env:myenv]
  upload_protocol = espota
  upload_port = IP_ADDRESS_HERE or mDNS_NAME.local
  ```

For example,

- `pio run -t upload --upload-port 192.168.0.255`
- `pio run -t upload --upload-port myesp8266.local`

**Authentication and upload options**

You can pass additional options/flags to OTA uploader using `upload_flags` option in "`platformio.ini`" (Project Configuration File)
Available flags

- `--port=ESP_PORT` ESP8266 OTA Port. Default 8266
- `--auth=AUTH` Set authentication password
- `--spiffs` Use this option to transmit a SPIFFS image and do not flash the module

For the full list with available options please run

```bash
~/.platformio/packages/framework-arduinoespressif8266/tools/espota.py --help
```

Usage: espota.py [options]

Transmit image over the air to the esp8266 module with OTA support.

Options:

- `-h, --help` show this help message and exit

Destination:

- `-i ESP_IP, --ip=ESP_IP` ESP8266 IP Address.
- `-I HOST_IP, --host_ip=HOST_IP` Host IP Address.
- `-p ESP_PORT, --port=ESP_PORT` ESP8266 ota Port. Default 8266
- `-P HOST_PORT, --host_port=HOST_PORT` Host server ota Port. Default random 10000-60000

Authentication:

- `-a AUTH, --auth=AUTH` Set authentication password.

Image:

- `-f FILE, --file=FILE` Image file.
- `-s, --spiffs` Use this option to transmit a SPIFFS image and do not flash the module.

Output:

- `-d, --debug` Show debug output. And override loglevel with debug.
- `-r, --progress` Show progress output. Does not work for ArduinoIDE
Demo

Using Arduino Framework with Staging version

PlatformIO will install the latest Arduino Core for ESP8266 from https://github.com/esp8266/Arduino. The Git should be installed in a system. To update Arduino Core to the latest revision, please open PlatformIO IDE and navigate to PlatformIO Home > Platforms > Updates.

1. Please install PlatformIO IDE

2. Initialize a new project, open “platformio.ini” (Project Configuration File) and specify the link to the framework repository in platform_packages section. For example,

   ```ini
   [env:nodemcuv2]
   platform = espressif8266
   board = nodemcuv2
   framework = arduino
   platform_packages =
   platformio/framework-arduinosoph8266 @ https://github.com/esp8266/Arduino.git
   ```

3. Try to build the project

4. If you see build errors, then try to build this project using the same stage with Arduino IDE

5. If it works with Arduino IDE but doesn’t work with PlatformIO, then please file a new issue with attached information:
   - test project/files
   - detailed log of build process from Arduino IDE (please copy it from console to https://hastebin.com)
   - detailed log of build process from PlatformIO Build System (please copy it from console to https://hastebin.com)
Examples

Examples are listed from Espressif 8266 development platform repository:

- arduino-webserver
- arduino-asyncudp
- arduino-blink
- simba-blink
- esp8266-RTOS-sdk-blink
- esp8266-nonos-sdk-blink
- arduino-wifiscan

Stable and upstream versions

You can switch between stable releases of Espressif 8266 development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

Stable

```ini
; Latest stable version
[env:latest_stable]
platform = espressif8266
board = ...

; Custom stable version
[env:custom_stable]
platform = espressif8266@x.y.z
board = ...
```

Upstream

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-espressif8266.git
board = ...
```
### Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinioespressif8266</td>
<td>Arduino Wiring-based Framework for Espressif ESP8266 microcontrollers</td>
</tr>
<tr>
<td>framework-esp8266-nonos-sdk</td>
<td>Espressif ESP8266 Non-OS SDK</td>
</tr>
<tr>
<td>framework-esp8266-rtos-sdk</td>
<td>Espressif ESP8266 SDK based on FreeRTOS</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba is an Embedded Programming Platform. It aims to make embedded program-</td>
</tr>
<tr>
<td></td>
<td>ming easy and portable</td>
</tr>
<tr>
<td>tool-esptool</td>
<td>Espressif ESP8266 build/flash helper tool</td>
</tr>
<tr>
<td>tool-esptoolpy</td>
<td>Espressif ESP8266 and ESP32 serial bootloader utility</td>
</tr>
<tr>
<td>tool-mklittlefs</td>
<td>Utility for creating littlefs images for upload on the ESP8266</td>
</tr>
<tr>
<td>tool-mkspiffs</td>
<td>Tool to build and unpack SPIFFS images</td>
</tr>
<tr>
<td>toolchain-xtensa</td>
<td>GCC Toolchain for Xtensa processor</td>
</tr>
</tbody>
</table>

**Warning: Linux Users:**

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

**Windows Users:**

Please check that you have a correctly installed USB driver from board manufacturer.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Arduino</em></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to con-</td>
</tr>
<tr>
<td></td>
<td>trol devices attached to a wide range of Arduino boards to create all kinds</td>
</tr>
<tr>
<td></td>
<td>of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><em>ESP8266 Non-OS SDK</em></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs)</td>
</tr>
</tbody>
</table>
## 4D Systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4D Systems gen4 IoT Range</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit HUZZAH ESP8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Amperka

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi Slot</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## DigiStump

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DigiStump Oak</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Doit

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Mx DevKit (ESP8285)</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPDuino (ESP-13 Module)</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## DycodeX

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPectro Core</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## ESPert

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPpresso Lite 1.0</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPpresso Lite 2.0</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
## ESPino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-WROOM-02</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif ESP8266 ESP-12E</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 1M</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 512k</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07 1MB</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07S</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Generic ESP8285 Module</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 1.0</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 2.0</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WifiInfo</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Heltec

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec Wifi kit 8</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## ITEAD

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonoff Basic</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff S20</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff SV</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff TH</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Invent One

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invent One</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU 0.9 (ESP-12 Module)</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>NodeMCU 1.0 (ESP-12E Module)</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olimex MOD-WIFI-ESP8266(-DEV)</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Schirmilabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schirmilabs Edulino WiFi</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wio Link</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Wio Node</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun Blynk Board</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing Dev</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### SweetPea

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SweetPea ESP-210</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### ThaiEasyElec

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThaiEasyElec ESPino</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### WEMOS

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS D1 R1</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos D1 R2 and mini</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos D1 mini Lite</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos D1 mini Pro</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>16MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
**WifiDuino**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFiduino</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

**XinaBox**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinaBox CW01</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

**Freescale Kinetis**

Configuration `platform = freescalekinetis`

Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

For more detailed information please visit vendor site.

---

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

---

**Examples**

Examples are listed from Freescale Kinetis development platform repository:

- zephyr-blink
- zephyr-net-telnet
- mbed-rtos-usb-msd
- zephyr-sensor-sx9500
- mbed-rtos-blink-baremetal
- mbed-rtos-ethernet-tls
- mbed-rtos-psa
- mbed-legacy-examples
- mbed-rtos-kvstore
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - **On-Board Debug Tools**
  - **External Debug Tools**

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>MK64FN11M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>MK20DX128VLL5</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>MK22FN512VLLH12</td>
<td>120MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>MK64FN11M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL0SZ</td>
<td>MKL05Z32VFM4</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>MKL25Z128VLLK4</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>MKL27Z64VLLH4</td>
<td>48MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>MKL43Z256VLLH4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>MKL46Z256VLL4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>MKW41Z512VHT4</td>
<td>48MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freescale Kinetis FRDM-KL82Z</td>
<td>MKL82Z128VLK7</td>
<td>96MHz</td>
<td>128KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Hexiwear</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>SEGGER IP Switch Board</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

**Stable and upstream versions**

You can switch between stable releases of Freescale Kinetis development platform and the latest upstream version using `platform` option in "platformio.ini" *(Project Configuration File)* as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = freescalekinetis
board = ...

; Custom stable version
[env:custom_stable]
platform = freescalekinetis@x.y.z
board = ...
```

**Upstream**

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-freescalekinetis.git
board = ...
```
# Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-nxp</td>
<td>NXP HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedtls module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUBoot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinybior</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of standard cryptography primitives for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARMv8-M and Zephyr framework</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-pyocd</td>
<td>Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP</td>
</tr>
<tr>
<td>toolchain-gccarmnoneeabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>
Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Boards

**Note:**

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### Freescale

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>On-board</td>
<td>MK20DX128VLH5</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>On-board</td>
<td>MK22FN512VLLH12</td>
<td>120MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>On-board</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>On-board</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>On-board</td>
<td>MKL05Z32VF4</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>On-board</td>
<td>MKL25Z128VLLK4</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>On-board</td>
<td>MKL27Z64VHLH4</td>
<td>48MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>On-board</td>
<td>MKL43Z256VLLH4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>On-board</td>
<td>MKL46Z256VLL4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL82Z</td>
<td>External</td>
<td>MKL82Z128VLK7</td>
<td>96MHz</td>
<td>128KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>External</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>On-board</td>
<td>MKW41Z512VHT4</td>
<td>48MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
MikroElektronika

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hexiwear</td>
<td>External</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

SEGGER

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGGER IP Switch Board</td>
<td>External</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

GigaDevice GD32V

**Configuration** `platform = gd32v`

The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

For more detailed information please visit vendor site.

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

**Examples**

Examples are listed from GigaDevice GD32V development platform repository:

- eval-blink
- arduino-blink
- longan-nano-blink

**Debugging**

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - External Debug Tools
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of GigaDevice GD32V development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

Stable

; Latest stable version
[env:latest_stable]
platform = gd32v
board = ...

; Custom stable version
[env:custom_stable]
platform = gd32v@x.y.z
board = ...

Upstream

[env:upstream_develop]
platform = https://github.com/sipeed/platform-gd32v.git
board = ...
Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduino-gd32v</td>
<td>Arduino Wiring-based Framework for GigaDevice GD32V microcontrollers</td>
</tr>
<tr>
<td>framework-gd32vf103-sdk</td>
<td>GigaDevice GD32VF103 Firmware Library</td>
</tr>
<tr>
<td>tool-dfuutil</td>
<td>Device Firmware Upgrade Utilities</td>
</tr>
<tr>
<td>tool-gd32vflash</td>
<td>GigaDevice GD32V Flash tools</td>
</tr>
<tr>
<td>tool-openocd-gd32v</td>
<td>Open On-Chip Debugger branch with RISC-V GigaDevice GD32V support</td>
</tr>
<tr>
<td>toolchain-gd32v</td>
<td>RISC-V GCC toolchain</td>
</tr>
</tbody>
</table>

Warning: Linux Users:

- Install “udev” rules 99-platformio-udev.rules
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
<tr>
<td>GD32V SDK</td>
<td></td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by pio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wio Lite RISC-V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
Sipeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>External</td>
<td>GD32VF103VB6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>External</td>
<td>GD32VF103CB6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>External</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Infineon XMC

Configuration \textit{platform} = \texttt{infineonxmc}

Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.

For more detailed information please visit \texttt{vendor site}.

Contents

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Examples

Examples are listed from Infineon XMC development platform repository:

- ifx9201
- device-control
- arduino-blink
- spi
- ultrasonic
- rtc
- radar
- arduino-wire

Debugging

\textit{Debugging} - “1-click” solution for debugging with a zero configuration.
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMC1100 Boot Kit</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>32KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>XMC1400</td>
<td>48MHz</td>
<td>1.95MB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>XMC4200</td>
<td>80MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>XMC4700</td>
<td>144MHz</td>
<td>2.00MB</td>
<td>1.95MB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Infineon XMC development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

```
; Latest stable version
[env:latest_stable]
platform = infineonxmc
board = ...

; Custom stable version
[env:custom_stable]
platform = infineonxmc@x.y.z
board = ...
```
Upstream

```
[env:upstream_develop]
platform = https://github.com/Infineon/platformio-infineonxmc.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinoxmc</td>
<td>Arduino Wiring-based Framework for Infineon’s XMC microcontrollers</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or [PlatformIO Boards Explorer](#).
- For more detailed board information please scroll the tables below by horizontally.
Infineon

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMC1100 Boot Kit</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>32KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>On-board</td>
<td>XMC1400</td>
<td>48MHz</td>
<td>1.95MB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>On-board</td>
<td>XMC4200</td>
<td>80MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>On-board</td>
<td>XMC4700</td>
<td>144MHz</td>
<td>2.00MB</td>
<td>1.95MB</td>
</tr>
</tbody>
</table>

Intel ARC32

Configuration $platform = intel_arc32$

ARC embedded processors are a family of 32-bit CPUs that are widely used in SoC devices for storage, home, mobile, automotive, and Internet of Things applications.

For more detailed information please visit vendor site.

Contents

- Examples
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Examples

Examples are listed from Intel ARC32 development platform repository:

- arduino-curie-imu
- arduino-blink
- arduino-internal-libs

Stable and upstream versions

You can switch between stable releases of Intel ARC32 development platform and the latest upstream version using $platform$ option in “platformio.ini” (Project Configuration File) as described below.
Stable

```plaintext
; Latest stable version
[env:latest_stable]
platform = intel_arc32
board = ...

; Custom stable version
[env:custom_stable]
platform = intel_arc32@x.y.z
board = ...
```

Upstream

```plaintext
[env:upstream_develop]
platform = https://github.com/platformio/platform-intel_arc32.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinointel</td>
<td>Arduino Wiring-based Framework Intel ARC32 processor</td>
</tr>
<tr>
<td>tool-arduino101load</td>
<td>Genuino101 uploader tool</td>
</tr>
<tr>
<td>toolchain-intelarc32</td>
<td>GCC for Intel ARC processor</td>
</tr>
</tbody>
</table>

**Warning: Linux Users:**

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](https://example.com)

**Windows Users:**

Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Boards

**Note:**

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer

1.10. Development Platforms
For more detailed board information please scroll the tables below by horizontally.

Intel

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino/Genuino 101</td>
<td>No</td>
<td>ARCV2EM</td>
<td>32MHz</td>
<td>152KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

**Intel MCS-51 (8051)**

Configuration `platform = intel_mcs51`

The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems. For more detailed information please visit vendor site.

Examples

Examples are listed from Intel MCS-51 (8051) development platform repository:

- stc-header
- native-blink
- stc-blink

Stable and upstream versions

You can switch between stable releases of Intel MCS-51 (8051) development platform and the latest upstream version using `platform` option in “`platformio.ini`” (Project Configuration File) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = intel_mcs51
board = ...

; Custom stable version
```
[env:custom_stable]
platform = intel_mcs51@x.y.z
board = ...

Upstream

[env:upstream_develop]
platform = https://github.com/platformio/platform-intel_mcs51.git
board = ...

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool-stcgal</td>
<td>Open Source STC MCU ISP flash tool</td>
</tr>
<tr>
<td>toolchain-sdcc</td>
<td>Small Device C compiler suite</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules 99-platformio-udev.rules
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
- Please check that you have a correctly installed USB driver from board manufacturer

Boards

Note:
- You can list pre-configured boards by pio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

Nuvoton

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic N79E8432</td>
<td>No</td>
<td>N79E8432</td>
<td>22MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic N79E844</td>
<td>No</td>
<td>N79E844</td>
<td>22MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic N79E845</td>
<td>No</td>
<td>N79E845</td>
<td>22MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic N79E854</td>
<td>No</td>
<td>N79E854</td>
<td>22MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic N79E855</td>
<td>No</td>
<td>N79E855</td>
<td>22MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
</tbody>
</table>
**STC**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic STC15F204EA</td>
<td>No</td>
<td>STC15F204EA</td>
<td>11MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic STC15F2K60S2</td>
<td>No</td>
<td>STC15F2K60S2</td>
<td>6MHz</td>
<td>60KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Generic STC15W204S</td>
<td>No</td>
<td>STC15W204S</td>
<td>11MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic STC15W404AS</td>
<td>No</td>
<td>STC15W404AS</td>
<td>11MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic STC15W408AS</td>
<td>No</td>
<td>STC15W408AS</td>
<td>11MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic STC89C52RC</td>
<td>No</td>
<td>STC89C52RC</td>
<td>11MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
</tbody>
</table>

**Kendryte K210**

Configuration  
\[\text{platform} = \text{kendryte210}\]

Kendryte K210 is an AI capable RISCV64 dual core SoC.

For more detailed information please visit vendor site.

---

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

---

**Examples**

Examples are listed from Kendryte K210 development platform repository:

- arduino-blink
- kendryte-standalone-sdk_hello
- kendryte-freertos-sdk_hello

---

**Debugging**

**Debugging** - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - External Debug Tools
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

External Debug Tools

Boards listed below are compatible with Debugging but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sipeed MAIX BiT</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Kendryte K210 development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = kendryte210
board = ...

; Custom stable version
[env:custom_stable]
platform = kendryte210@x.y.z
board = ...
```

**Upstream**

```ini
[env:upstream_develop]
platform = https://github.com/sipeed/platform-kendryte210.git
board = ...
```
Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-kendryte-freertos-sdk</td>
<td>Kendryte K210 SDK with FreeRTOS</td>
</tr>
<tr>
<td>framework-kendryte-standalone-sdk</td>
<td>Kendryte K210 standalone SDK without OS support</td>
</tr>
<tr>
<td>framework-maixduino</td>
<td>Arduino ore for Maix Board (K210)</td>
</tr>
<tr>
<td>tool-kflash-kendryte210</td>
<td>A Python-based Kendryte K210 UART ISP Utility</td>
</tr>
<tr>
<td>tool-openocd-kendrye</td>
<td>Open On-Chip Debugger branch with RISC-V Kendryte support</td>
</tr>
<tr>
<td>toolchain-kendryte210</td>
<td>RISC-V GCC Toolchain for Kendryte 210</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
- Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
Sipeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sipeed MAIX BiT</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>

Lattice iCE40

**Configuration**  
`platform = lattice_ice40`

The iCE40 family of ultra-low power, non-volatile FPGAs has five devices with densities ranging from 384 to 7680 Look-Up Tables (LUTs). In addition to LUT-based, low-cost programmable logic, these devices feature Embedded Block RAM (EBR), Non-volatile Configuration Memory (NVCM) and Phase Locked Loops (PLLs). These features allow the devices to be used in low-cost, high-volume consumer and system applications.

For more detailed information please visit [vendor site](#).

**Contents**

- Examples
- Stable and upstream versions
- Packages
- Boards

**Examples**

Examples are listed from Lattice iCE40 development platform repository:

- leds
- counter

**Stable and upstream versions**

You can switch between stable releases of Lattice iCE40 development platform and the latest upstream version using `platform` option in “`platformio.ini`” (Project Configuration File) as described below.

**Stable**

```
; Latest stable version
[env:latest_stable]
platform = lattice_ice40
board = ...
```
Custom stable version

```
[env:custom_stable]
platform = lattice_ice40@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-lattice_ice40.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolchain-icestorm</td>
<td>Tools for analyzing and creating bitstream files for FPGA IceStorm</td>
</tr>
<tr>
<td>toolchain-iverilog</td>
<td>Verilog simulation and synthesis tool</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

Windows Users:

- Please check that you have a correctly installed USB driver from board manufacturer

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed `board` information please scroll the tables below by horizontally.

FPGAs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IceZUM Alhambra FPGA</td>
<td>No</td>
<td>ICE40-HX1K-TQ144</td>
<td>12MHz</td>
<td>32KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
Lattice

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice iCEstick FPGA Evaluation Kit</td>
<td>No</td>
<td>ICE40-HX1K-TQ144</td>
<td>12MHz</td>
<td>32KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Maxim 32

**Configuration** \( \text{platform} = \text{maxim32} \)

Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

For more detailed information please visit [vendor site](#).

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

**Examples**

Examples are listed from Maxim 32 development platform repository:

- mbed-rtos-mutex
- mbed-rtos-blink-baremetal
- mbed-rtos-sysinfo
- mbed-legacy-examples

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maxim ARM mbed Enabled Development Platform for MAX32600</td>
<td>MAX32600</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX32620FTHR</td>
<td>MAX32620FTHR</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Maxim Health Sensor Platform</td>
<td>MAX32620</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Maxim Wireless Sensor Node Demonstrator</td>
<td>MAX32610</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Maxim 32 development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

Stable

```
; Latest stable version
[env:latest_stable]
platform = maxim32
board = ...

; Custom stable version
[env:custom_stable]
platform = maxim32@x.y.z
board = ...
```
Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-maxim32.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-pyocd</td>
<td>Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
Maxim

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX32620FTHR</td>
<td>External</td>
<td>MAX32620FTHR</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>MAX32625MBED</td>
<td>No</td>
<td>MAX32625</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
<tr>
<td>MAX32625NEXPAQ</td>
<td>No</td>
<td>MAX32625</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
<tr>
<td>MAX32625PICO</td>
<td>No</td>
<td>MAX32625</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
<tr>
<td>Maxim ARM mbed Enabled Development Platform for MAX32600</td>
<td>On-board</td>
<td>MAX32600</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Maxim Health Sensor Platform</td>
<td>External</td>
<td>MAX32620</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Maxim MAX32630FTHR Application Platform</td>
<td>No</td>
<td>MAX32630</td>
<td>96MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Maxim Wireless Sensor Node Demonstrator</td>
<td>External</td>
<td>MAX32610</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Sigma Delta Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT32620B</td>
<td>No</td>
<td>MAX32620IWG</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>SDT32625B</td>
<td>No</td>
<td>MAX32625ITK</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
</tbody>
</table>

Microchip PIC32

Configuration `platform = microchippic32`

Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

For more detailed information please visit vendor site.

Contents

- Examples
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Examples

Examples are listed from Microchip PIC32 development platform repository:

- arduino-blink
• arduino-internal-libs

Stable and upstream versions

You can switch between stable releases of Microchip PIC32 development platform and the latest upstream version using platform option in "platformio.ini" (Project Configuration File) as described below.

Stable

```
; Latest stable version
[env:latest_stable]
platform = microchippic32
board = ...

; Custom stable version
[env:custom_stable]
platform = microchippic32@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-microchippic32.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinomicrochippic32</td>
<td>Arduino Wiring-based Framework for Microchip PIC32 microcontrollers</td>
</tr>
<tr>
<td>tool-pic32prog</td>
<td>Flash programming utility for Microchip PIC32 microcontrollers</td>
</tr>
<tr>
<td>toolchain-microchippic32</td>
<td>GCC Toolchain for Microchip PIC32</td>
</tr>
</tbody>
</table>

**Warning: Linux Users:**

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

**Windows Users:**

Please check that you have a correctly installed USB driver from board manufacturer
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

4D Systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4D Systems PICadillo 35T</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

BOXTEC

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelvePic32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>48MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HelvePic32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>48MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HelvePic32 Robot</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>244KB</td>
<td>62KB</td>
</tr>
<tr>
<td>HelvePic32 SMD MX270</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>244KB</td>
<td>62KB</td>
</tr>
</tbody>
</table>

ChipKIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB Station</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>240KB</td>
<td>62KB</td>
</tr>
</tbody>
</table>
### Digilent

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digilent Cerebot 32MX4</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent Cerebot 32MX7</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent OpenScope</td>
<td>No</td>
<td>32MZ2048EFG124</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Digilent chipKIT Cmod</td>
<td>No</td>
<td>32MX150F128D</td>
<td>40MHz</td>
<td>124KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent chipKIT DP32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent chipKIT MAX32</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent chipKIT MX3</td>
<td>No</td>
<td>32MX320F128H</td>
<td>80MHz</td>
<td>124KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Digilent chipKIT Pro MX4</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent chipKIT Pro MX7</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent chipKIT UNO32</td>
<td>No</td>
<td>32MX320F128H</td>
<td>80MHz</td>
<td>124KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Digilent chipKIT WF32</td>
<td>No</td>
<td>32MX695F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent chipKIT WiFire</td>
<td>No</td>
<td>32MZ2048EFG100</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Digilent chipKIT uC32</td>
<td>No</td>
<td>32MX340F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>chipKIT WiFire rev. C</td>
<td>No</td>
<td>32MZ2048EFG100</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

### Fubarino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fubarino Mini</td>
<td>No</td>
<td>32MX250F128D</td>
<td>48MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Fubarino SD (1.5)</td>
<td>No</td>
<td>32MX795F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Mini 2.0</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>240KB</td>
<td>62KB</td>
</tr>
</tbody>
</table>

### Makerology

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStation Mini</td>
<td>No</td>
<td>32MX150F128C</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### MikroElektronika

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MikroElektronika Clicker 2</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>MikroElektronika Flip N Click MZ</td>
<td>No</td>
<td>32MZ2048EFH100</td>
<td>252MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olimex PIC32-PINGUINO</td>
<td>No</td>
<td>32MX440F256H</td>
<td>80MHz</td>
<td>252KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
## OpenBCI

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenBCI 32bit</td>
<td>No</td>
<td>32MX250F128B</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## PONTECH

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PONTECH UAV100</td>
<td>No</td>
<td>32MX440F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Pontech

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pontech NoFire</td>
<td>No</td>
<td>32MZ2048EFG100</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Pontech Quick240</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeeedStudio CUI32stem</td>
<td>No</td>
<td>32MX795F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pic32 CUI32-Development Stick</td>
<td>No</td>
<td>32MX440F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## UBW32

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBW32 MX460</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>UBW32 MX795</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## chipKIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>chipKIT Lenny</td>
<td>No</td>
<td>32MX270F256D</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
### Nordic nRF51

**Configuration** $platform = \text{nordicnrf51}$

The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

For more detailed information please visit vendor site.

### Examples

Examples are listed from Nordic nRF51 development platform repository:

- mbed-blink
- mbed-serial
- zephyr-blink
- mbed-events
- arduino-blink
- arduino-ble-led
- zephyr-ble-eddystone
- arduino-internal-libs
- zephyr-drivers-entropy
- mbed-ble-thermometer

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

### On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Calliope mini</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Delta DFCM-NNN40</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>JKSoft Wallbot BLE</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Arch BLE</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Switch Science mbed HRM1017</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Switch Science mbed TY51822r3</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>VNG VBLUNOSi1</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>y3 nRF51822 mbug</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BluzDK</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>OSHChip</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sino:Bit</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ng-beacon</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**Stable and upstream versions**

You can switch between stable releases of Nordic nRF51 development platform and the latest upstream version using `platform` option in “`platformio.ini`” (Project Configuration File) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = nordicnrf51
board = ...

; Custom stable version
[env:custom_stable]
platform = nordicnrf51@x.y.z
board = ...
```

**Upstream**

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-nordicnrf51.git
board = ...
```

**Packages**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinonordicnrf5</td>
<td>Arduino Wiring-based Framework for Nordic Semiconductor nRF5 based boards</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-nordic</td>
<td>Nordic nRF5x HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
</tbody>
</table>

1.10. Development Platforms
### Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinycbor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARM</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-nrfjprog</td>
<td>nRFx command line tools</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and</td>
</tr>
<tr>
<td>tool-sreccat</td>
<td>Collection of powerful tools for manipulating EPROM load files</td>
</tr>
<tr>
<td>toolchain-gccarmnoneeabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

#### Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

#### Windows Users:
Please check that you have a correctly installed USB driver from board manufacturer

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td><strong>Arduino</strong> Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td><strong>Mbed</strong> Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr</td>
<td><strong>Zephyr</strong> The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Boards

**Note:**
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
For more detailed board information please scroll the tables below by horizontally.

### BBC

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### BluzDK

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BluzDK</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Calliope

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calliope mini</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Delta

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta DFCM-NNN40</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### JKSoft

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>JKSoft Wallbot BLE</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Nordic

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit(PCA1000X)</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
**OSHChip**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OSHChip</strong></td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**RedBearLab**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RedBearLab BLE Nano 1.5</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td><strong>RedBearLab nRF51822</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

**SeeedStudio**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Seeed Arch BLE</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td><strong>Seeed Arch Link</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td><strong>Seeed Tiny BLE</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

**Switch Science**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Switch Science mbed HRM1017</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td><strong>Switch Science mbed TY51822r3</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**VNG**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VNG VBLUNO51</strong></td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**Waveshare**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Waveshare BLE400</strong></td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**ng-beacon**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ng-beacon</strong></td>
<td>External</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
sino:bit

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sino:Bit</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

y5 design

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>y5 NRF51822 mbug</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

**Nordic nRF52**

Configuration `platform = nordicnrf52`

The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

For more detailed information please visit vendor site.

### Contents

- **Tutorials**
- **Examples**
- **Debugging**
- **Stable and upstream versions**
- **Packages**
- **Frameworks**
- **Boards**

### Tutorials

- Arduino and Nordic nRF52-DK: debugging and unit testing
- Zephyr and Nordic nRF52-DK: debugging, unit testing, project analysis

### Examples

Examples are listed from Nordic nRF52 development platform repository:

- zephyr-blink
- zephyr-net-echo-client
- arduino-blink
- arduino-ble-led
- mbed-rtos-blink-baremetal
- zephyr-ble-beacon
- zephyr-subsys-nvs
- mbed-rtos-nfc
- mbed-legacy-examples
- arduino-nina-b1-generic-example
- arduino-bluefruit-bleuart
- arduino-serial-plotter
- mbed-rtos-ble-thermometer

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools

**Tools & Debug Probes**

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

**On-Board Debug Tools**

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit V2</td>
<td>NRF52833</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BL652 Development Kit</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ElectronutLabs Blip</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ElectronutLabs Papyr</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>reel_board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

**External Debug Tools**

Boards listed below are compatible with Debugging but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Nitrogen</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Arduíno Nano 33 BLE</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>960KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Holyiot YJ-16019</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ItsyBitsy nRF52840 Express</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Particle Argon</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Boron</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Xenon</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Raytac MDBG1Q-RX Dongle</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Ruuvi Tag</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>SDTS2832B</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
Stable and upstream versions

You can switch between stable releases of Nordic nRF52 development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = nordicnrf52
board = ...

; Custom stable version
[env:custom_stable]
platform = nordicnrf52@x.y.z
board = ...
```

**Upstream**

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-nordicnrf52.git
board = ...
```

**Packages**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduino-nrf52-mbedos</td>
<td>Arduino framework supporting mbed-enabled boards</td>
</tr>
<tr>
<td>framework-arduino-adafritunrf52</td>
<td>Arduino Wiring-based Framework for Nordic Semiconductor nRF52 BLE SoC</td>
</tr>
<tr>
<td>framework-arduino-nordicnrf5</td>
<td>Arduino Wiring-based Framework for Nordic Semiconductor nRF5 based boards</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-nordic</td>
<td>Nordic nRF5x HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>framework-zephyr-tinycbor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of standard cryptography primitives for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARM Cortex-M processors</td>
</tr>
<tr>
<td>tool-bossac-nordicnrf52</td>
<td>Basic Open Source SAM-BA Application (BOSSA) for Nordic nRF52 microcontrollers</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-nrfjprog</td>
<td>nRFx command line tools</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and Debugging</td>
</tr>
<tr>
<td>tool-sreccat</td>
<td>Collection of powerful tools for manipulating EPROM load files</td>
</tr>
<tr>
<td>toolchain-gccarmnoneeabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

**Warning: Linux Users:**
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](https://www.platformio.org/docs/boards/raspberry-pi.html).

**Windows Users:**
Please check that you have a correctly installed USB driver from board manufacturer

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Boards

**Note:**
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
## 96Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Nitrogen</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>ItsyBitsy nRF52840 Express</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

## Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Nano 33 BLE</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>960KB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

## BBC

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit V2</td>
<td>On-board</td>
<td>NRF52833</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Delta

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta DFBM-NQ620</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Electronut Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
ElectronutLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElectronutLabs Blip</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ElectronutLabs Papyr</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Holyiot

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holyiot YJ-16019</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

Laird Connectivity

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL652 Development Kit</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Makerdiary

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Nordic

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

PHYTEC

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>reel_board</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
## Particle

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Argon</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Boron</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Xenon</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

## Raytac

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raytac MDBT50Q-RX Dongle</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

## RedBearLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Ruuvi

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruvi Tag</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Sigma Delta Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT52832B</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Taida Century

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## VNG

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNG VBLUno52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
**decaWave**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

**u-blox**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

**Nuclei**

Configuration _platform_ = _nuclei_

Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.

For more detailed information please visit vendor site.

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

**Examples**

Examples are listed from Nuclei development platform repository:

- freertos_demo
- rtthread_demo
- demo_dsp
- helloworld
- dhrystone
- whetstone
- demo_eclic
- ucosii_demo
- coremark
- demo_timer

1.10. Development Platforms 455
Debugging

- rtthread_msh
- demo_nice

Debugging - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HummingBird Evaluation Kit</td>
<td>HUMMINGBIRD</td>
<td>5MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Nuclei development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.
Stable

```plaintext
; Latest stable version
[env:latest_stable]
platform = nuclei
board = ...

; Custom stable version
[env:custom_stable]
platform = nuclei@x.y.z
board = ...
```

Upstream

```plaintext
[env:upstream_develop]
platform = https://github.com/Nuclei-Software/platform-nuclei.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-nuclei-sdk</td>
<td>Nuclei N/NX Embedded Software Development Kit</td>
</tr>
<tr>
<td>tool-openocd-nuclei</td>
<td>Open On-Chip Debugger branch with RISC-V Nuclei support</td>
</tr>
<tr>
<td>toolchain-riscv-gcc-nuclei</td>
<td>Nuclei RISC-V GCC toolchain</td>
</tr>
</tbody>
</table>

Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#).

Warning: Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nuclei SDK</strong></td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
For more detailed board information please scroll the tables below by horizontally.

### GigaDevice

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Nuclei

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>On-board</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HummingBird Evaluation Kit</td>
<td>On-board</td>
<td>HUMMINGBIRD</td>
<td>5MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### NXP i.MX RT

**Configuration** `platform = nxpimxrt`

The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.

For more detailed information please visit vendor site.

### Contents

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

### Examples

Examples are listed from NXP i.MX RT development platform repository:

- zephyr-blink
- mbed-rtos-blink-baremetal
- mbed-rtos-cellular
- mbed-rtos-ethernet-tls
- zephyr-synchronization
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - **On-Board Debug Tools**

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXP i.MX RT1010 Evaluation Kit</td>
<td>MIMXRT1011DAE5A</td>
<td>500MHz</td>
<td>64KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1015 Evaluation Kit</td>
<td>MIMXRT1015DAF5A</td>
<td>500MHz</td>
<td>96KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1020 Evaluation Kit</td>
<td>MIMXRT1021DAG5A</td>
<td>500MHz</td>
<td>64MB</td>
<td>256MB</td>
</tr>
<tr>
<td>NXP i.MX RT1050 Evaluation Kit</td>
<td>MIMXRT1052DVL6B</td>
<td>600MHz</td>
<td>64MB</td>
<td>512KB</td>
</tr>
<tr>
<td>NXP i.MX RT1060 Evaluation Kit</td>
<td>MIMXRT1062DVL6A</td>
<td>600MHz</td>
<td>64MB</td>
<td>1MB</td>
</tr>
<tr>
<td>NXP i.MX RT1064 Evaluation Kit</td>
<td>MIMXRT1064DVL6A</td>
<td>600MHz</td>
<td>4MB</td>
<td>1MB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of NXP i.MX RT development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = nxpimxrt
board = ...

; Custom stable version
[env:custom_stable]
platform = nxpimxrt@x.y.z
board = ...
```

1.10. Development Platforms
Upstream

[env:upstream_develop]
platform = https://github.com/platformio/platform-nxpmxrt.git
board = ...

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-nxp</td>
<td>NXP HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinybor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinyhttps</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of cryptographic primitives for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-tinyhttps</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of cryptographic primitives for Zephyr framework</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package programs</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and Debugging (OpenChIP)</td>
</tr>
<tr>
<td>tool-pyoced</td>
<td>Open source python library for programming and debugging ARM Cortex-M microcontroller</td>
</tr>
<tr>
<td>toolchain-gccarmnoneeabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

Warning: Linux Users:

- Install “udev” rules 99-platformio-udev.rules
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

NXP

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXP i.MX RT1010 Evaluation Kit</td>
<td>On-board</td>
<td>MIMXRT1011DAE5A</td>
<td>500MHz</td>
<td>64KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1015 Evaluation Kit</td>
<td>On-board</td>
<td>MIMXRT1015DAF5A</td>
<td>500MHz</td>
<td>96KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1020 Evaluation Kit</td>
<td>On-board</td>
<td>MIMXRT1021DAG5A</td>
<td>500MHz</td>
<td>64MB</td>
<td>256MB</td>
</tr>
<tr>
<td>NXP i.MX RT1050 Evaluation Kit</td>
<td>On-board</td>
<td>MIMXRT1052DVL6B</td>
<td>600MHz</td>
<td>64MB</td>
<td>512KB</td>
</tr>
<tr>
<td>NXP i.MX RT1060 Evaluation Kit</td>
<td>On-board</td>
<td>MIMXRT1062DVL6A</td>
<td>600MHz</td>
<td>64MB</td>
<td>1MB</td>
</tr>
<tr>
<td>NXP i.MX RT1064 Evaluation Kit</td>
<td>On-board</td>
<td>MIMXRT1064DVL6A</td>
<td>600MHz</td>
<td>4MB</td>
<td>1MB</td>
</tr>
</tbody>
</table>

NXP LPC

**Configuration** `platform = nxplpc`

The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

For more detailed information please visit vendor site.

Contents

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks

1.10. Development Platforms
Examples

Examples are listed from NXP LPC development platform repository:

- zephyr-blink
- mbed-rtos-blockdevice
- mbed-rtos-blink-baremetal
- mbed-rtos-thread-statistics
- zephyr-custom-board
- mbed-legacy-examples
- zephyr-synchronization

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.
# External Debug Tools

Boards listed below are compatible with Debugging but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>DipCortex M3</td>
<td>LPC1347</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11C24</td>
<td>LPC11C24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U34</td>
<td>LPC11U34</td>
<td>48MHz</td>
<td>40KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U37</td>
<td>LPC11U37</td>
<td>48MHz</td>
<td>128KB</td>
<td>10KB</td>
</tr>
<tr>
<td>NXP LPCXpresso1549</td>
<td>LPC1549</td>
<td>72MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>LPC11U24</td>
<td>50MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>y5 LPC11U35 mbug</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

# Stable and upstream versions

You can switch between **stable releases** of NXP LPC development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

## Stable

```ini
; Latest stable version
[env:latest_stable]
platform = nxplpc
board = ...
```

(continues on next page)
### Custom stable version

```
[env:custom_stable]
platform = nxplpc@x.y.z
board = ...
```

### Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-nxplpc.git
board = ...
```

### Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-nxp</td>
<td>NXP HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-sgger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinyocbor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinyctrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of cryptographic primitives</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARM Cortex-M microcontrollers</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software programs</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and In-System Testing</td>
</tr>
<tr>
<td>tool-pyocd</td>
<td>Open source python library for programming and debugging ARM Cortex-M microcontrollers</td>
</tr>
<tr>
<td>toolchain-gccarmnoneeabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>
Warning: Linux Users:
- Install “udev” rules 99-platformio-udev.rules
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephy RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by pio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

AppNearMe

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroNFCBoard</td>
<td>No</td>
<td>LPC11U34</td>
<td>48MHz</td>
<td>48KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

CQ Publishing

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

Elektor Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoCo-ri-Co!</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>
## Embedded Artists

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

## GHI Electronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBuino</td>
<td>No</td>
<td>LPC11U24</td>
<td>30MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

## Micromint

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bambino-210E</td>
<td>On-board</td>
<td>LPC4330</td>
<td>204MHz</td>
<td>8MB</td>
<td>264KB</td>
</tr>
</tbody>
</table>

## NGX Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>External</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

## NXP

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>On-board</td>
<td>LPC11U68</td>
<td>50MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11C24</td>
<td>External</td>
<td>LPC11C24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U34</td>
<td>External</td>
<td>LPC11U34</td>
<td>48MHz</td>
<td>40KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U37</td>
<td>External</td>
<td>LPC11U37</td>
<td>48MHz</td>
<td>128KB</td>
<td>10KB</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>NXP LPCXpresso1549</td>
<td>External</td>
<td>LPC1549</td>
<td>72MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>On-board</td>
<td>LPC54114J256BD64</td>
<td>100MHz</td>
<td>256KB</td>
<td>192KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54608</td>
<td>On-board</td>
<td>LPC54608ET512</td>
<td>180MHz</td>
<td>512KB</td>
<td>200KB</td>
</tr>
<tr>
<td>NXP LPCXpresso55S16</td>
<td>On-board</td>
<td>LPC55S16</td>
<td>150MHz</td>
<td>256KB</td>
<td>96KB</td>
</tr>
<tr>
<td>NXP LPCXpresso55S69</td>
<td>On-board</td>
<td>LPC55S69</td>
<td>150MHz</td>
<td>640KB</td>
<td>320KB</td>
</tr>
<tr>
<td>NXP mbed LPC11U24</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP mbed LPC1768</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

466 Chapter 1. Contents
## Outrageous Circuits

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outrageous Circuits mBuino</td>
<td>No</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

## SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Arch GPRS V2</td>
<td>No</td>
<td>LPC11U37</td>
<td>48MHz</td>
<td>128KB</td>
<td>10KB</td>
</tr>
<tr>
<td>Seeed Arch Pro</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Seeed Xadow M0</td>
<td>No</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

## Smeshlink

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smeshlink xbed LPC1768</td>
<td>No</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Solder Splash Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DipCortex M3</td>
<td>External</td>
<td>LPC1347</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>External</td>
<td>LPC11U24</td>
<td>50MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

## Switch Science

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>On-board</td>
<td>LPC1114FN28</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

## u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-blox C027</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## y5 design

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>y5 LPC11U35 mbug</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>
RISC-V GAP

**Configuration**  
`platform = riscv_gap`

GreenWaves GAP8 IoT application processor enables the cost-effective development, deployment and autonomous operation of intelligent sensing devices that capture, analyze, classify and act on the fusion of rich data sources such as images, sounds or vibrations.

For more detailed information please visit vendor site.

**Contents**

- Configuration
- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

**Configuration**

- Drivers
- AutoTiler
- Running modes
  - Run from RAM
  - Run from RAM (without any bridge interaction)
  - Flash and run from RAM
  - Flash and run from Flash
  - Run from Flash
  - Run from Flash (without any bridge interaction)
- Uploading files to HyperFlash

**Drivers**

See “Drivers” section for *FTDI Chip* debug probe.

**AutoTiler**

You need GAP8 AutoTiler library, please request it via support@greenwaves-technologies.com
Put a library somewhere on a disk and add this folder to library path using `build_flags` in `platformio.ini` (Project Configuration File). For example,

```
[env:gapuino]
platform = riscv_gap
board = gapuino
framework = ...
build_flags = -L/path/to/libtile/folder
```

## Running modes

GAPuino supports 2 main modes:

1. Running from RAM, `boot_mode=jtag`
2. Running from HyperFlash, `boot_mode=jtag_hyper`

A running process can be controlled through the internal upload commands:

- `load`, @TODO
- `reqloop`, @TODO
- `ioloop`, @TODO
- `start`, @TODO
- `wait`, @TODO

You can configure “boot mode” and list of upload commands using “platformio.ini” (Project Configuration File). Default values are:

- `board_upload.boot_mode = jtag`
- `board_upload.commands = load reqloop ioloop start wait`

## Run from RAM

This is a default behavior when you run “Upload” task in PlatformIO IDE or use PlatformIO Core (CLI) and `pio run --target` command with `upload` target.

## Run from RAM (without any bridge interaction)

- Configure build environment using “platformio.ini” (Project Configuration File) as described below

```
[env:gapuino]
platform = riscv_gap
board = gapuino
framework = ...
board_upload.commands = load start
```

- Run “Upload” task in PlatformIO IDE or use PlatformIO Core (CLI) and `pio run --target` command with `upload` target.

## Flash and run from RAM

The same as Uploading files to HyperFlash.
Flash and run from Flash

- Configure build environment using "platformio.ini" (Project Configuration File) as described below

```ini
[env:gapuino]
platform = riscv_gap
board = gapuino
framework = ...
board_upload.boot_mode = jtag_hyper
board_upload.commands = reqloop ioloop start wait
```

- Perform *Uploading files to HyperFlash*.

Run from Flash

**Note:** You have to perform *Uploading files to HyperFlash* before.

- Configure build environment using "platformio.ini" (Project Configuration File) as described below

```ini
[env:gapuino]
platform = riscv_gap
board = gapuino
framework = ...
board_upload.boot_mode = jtag_hyper
board_upload.commands = start
```

- Run “Upload” task in *PlatformIO IDE* or use *PlatformIO Core (CLI)* and `pio run --target` command with upload target.

Run from Flash (without any bridge interaction)

**Note:** You have to perform *Uploading files to HyperFlash* before.

- Configure build environment using "platformio.ini" (Project Configuration File) as described below

```ini
[env:gapuino]
platform = riscv_gap
board = gapuino
framework = ...
board_upload.boot_mode = jtag_hyper
board_upload.commands = reqloop ioloop start wait
```

- Run “Upload” task in *PlatformIO IDE* or use *PlatformIO Core (CLI)* and `pio run --target` command with upload target.

Uploading files to HyperFlash

1. Create new project using *PlatformIO IDE* or initialize project using *PlatformIO Core (CLI)* and `pio project init` (if you have not initialized it yet)
2. Create a data folder (it should be on the same level as src folder) and put files here. Also, you can specify own location for data_dir.

3. Run “Upload File System image” task in PlatformIO IDE or use PlatformIO Core (CLI) and pio run --target command with uploadfs target.

Examples:

- PULP OS File System

## Examples

Examples are listed from RISC-V GAP development platform repository:

- gapuino-mbed-os-irq
- gapuino-mbed-autotiler-cifar10
- gapuino-mbed-driver-hyper-flash
- gapuino-pulp-os-i2c-eeprom
- gapuino-mbed-events-queue
- gapuino-pulp-os-kernel-dma
- gapuino-mbed-driver-hyper-rtc-alarm
- gapuino-mbed-fft2d
- gapuino-pulp-os-autotiler-bilinear-resize
- gapuino-mbed-matadd
- gapuino-mbed-features-cluster-dma
- gapuino-mbed-features-filesystem
- gapuino-pulp-os-filesystem
- gapuino-mbed-os-memory-pool
- gapuino-pulp-os-autotiler-cifar10
- gapuino-mbed-driver-cpp-raw-serial
- gapuino-pulp-os-hello-world

## Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAPduino GAP8</td>
<td>GAP8</td>
<td>250MHz</td>
<td>64MB</td>
<td>8MB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of RISC-V GAP development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = riscv_gap
board = ...

; Custom stable version
[env:custom_stable]
platform = riscv_gap@x.y.z
board = ...
```

**Upstream**

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-riscv_gap.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-gap_sdk</td>
<td>SDK for Greenwaves Technologies’ GAP8 IoT Application Processor</td>
</tr>
<tr>
<td>tool-pulp_tools</td>
<td>Top project for building PULP development tools</td>
</tr>
<tr>
<td>toolchain-riscv-pulp</td>
<td>GNU toolchain for RISC-V (PULP platform)</td>
</tr>
</tbody>
</table>
Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#)

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer

---

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td><strong>PULP OS</strong></td>
<td>PULP is a silicon-proven Parallel Ultra Low Power platform targeting high energy efficiencies. The platform is organized in clusters of RISC-V cores that share a tightly-coupled data memory</td>
</tr>
</tbody>
</table>

---

## Boards

**Note:**

- You can list pre-configured boards by `pio boards` command or [PlatformIO Boards Explorer](#)
- For more detailed board information please scroll the tables below by horizontally.

### GreenWaves Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAPuino GAP8</td>
<td>On-board</td>
<td>GAP8</td>
<td>250MHz</td>
<td>64MB</td>
<td>8MB</td>
</tr>
</tbody>
</table>

### Shakti

**Configuration** `platform = shakti`

Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

For more detailed information please visit [vendor site](#).

---

**Contents**

- Examples
- Debugging

---

1.10. Development Platforms
Examples

Examples are listed from Shakti development platform repository:

- shakti-sdk_gpio-keypad
- shakti-sdk_i2c-lm75
- shakti-sdk_uart-hello
- shakti-sdk_weatherstation

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Arty A7-100: Artix-7 FPGA Development Board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
<tr>
<td>Parashu on Artix-7 100T Arty FPGA Evaluation Kit</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Pinaka on Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Vajra on Arty A7-100: Artix-7 FPGA Development Board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
</tbody>
</table>
Stable and upstream versions

You can switch between stable releases of Shakti development platform and the latest upstream version using `platform` option in "platformio.ini" (Project Configuration File) as described below.

Stable

```ini
; Latest stable version
[env:latest_stable]
platform = shakti
board = ...

; Custom stable version
[env:custom_stable]
platform = shakti@x.y.z
board = ...
```

Upstream

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-shakti.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-shakti-sdk</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-openocd-riscv</td>
<td>Fork of Open On-Chip Debugger that has RISC-V support</td>
</tr>
<tr>
<td>tool-qemu-riscv</td>
<td>QEMU is a generic and open source machine emulator and virtualizer</td>
</tr>
<tr>
<td>toolchain-riscv</td>
<td>GNU toolchain for RISC-V, including GCC</td>
</tr>
</tbody>
</table>

Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#)

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakti SDK</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
</tbody>
</table>
Boards

---

**Note:**
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### Xilinx

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Arty A7-100: Artix-7 FPGA Development Board</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
<tr>
<td>Parashu on Artix-7 100T Arty FPGA Evaluation Kit</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Pinaka on Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Vajra on Arty A7-100: Artix-7 FPGA Development Board</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
</tbody>
</table>

### SiFive

**Configuration** `platform = sifive`

SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.

For more detailed information please visit vendor site.

---

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

**Examples**

Examples are listed from SiFive development platform repository:
- `freedom-e-sdk_freertos-blinky-system-view`
• zephyr-blink
• native-blink_asm
• freedom-e-sdk_hello
• freedom-e-sdk_test-coreip
• freedom-e-sdk_sifive-welcome
• freedom-e-sdk_timer-interrupt
• freedom-e-sdk_freertos-blinky
• freedom-e-sdk_multicore-hello
• freedom-e-sdk_user-syscall
• freedom-e-sdk_spi
• freedom-e-sdk_freertos-pmp-blinky
• freedom-e-sdk_user-mode
• zephyr-synchronization
• zephyr-hello-world

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of SiFive development platform and the latest upstream version using `platform` option in “`platformio.ini` (Project Configuration File)” as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = sifive
board = ...

; Custom stable version
[env:custom_stable]
platform = sifive@x.y.z
board = ...
```

**Upstream**

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-sifive.git
board = ...
```
## Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-freedom-e-sdk</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-fats</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-lorawan-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinycbor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of standard cryptography primitives for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARMv8-M and Zephyr framework</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-openocd-riscv</td>
<td>Fork of Open On-Chip Debugger that has RISC-V support</td>
</tr>
<tr>
<td>tool-qemu-riscv</td>
<td>QEMU is a generic and open source machine emulator and virtualizer</td>
</tr>
<tr>
<td>tool-renode</td>
<td>Renode is a development framework which accelerates IoT and embedded systems development by letting you simulate physical hardware systems</td>
</tr>
<tr>
<td>toolchain-riscv</td>
<td>GNU toolchain for RISC-V, including GCC</td>
</tr>
</tbody>
</table>
Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:

Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

SiFive

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiFive Unleashed</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>
Xilinx

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
</tbody>
</table>

Silicon Labs EFM32

Configuration `platform = siliconlabsefm32`

Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

For more detailed information please visit vendor site.

Examples

Examples are listed from Silicon Labs EFM32 development platform repository:

- zephyr-subsys-console-getline
- zephyr-blink
- mbed-rtos-blink-baremetal
- mbed-rtos-usb-cdc
- mbed-rtos-semaphore
- zephyr-custom-module
- mbed-legacy-examples
- zephyr-sensor-vl53l0x

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>EFM32GG990F1024</td>
<td>48MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>EFM32LG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>EFM32WG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>EFM32ZG222F32</td>
<td>24MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SLSTK3401A Pearl Gecko PG1</td>
<td>EFM32PG1B200F256GM48</td>
<td>40MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SLSTK3701A Giant Gecko SI</td>
<td>EFM32GG11B820F2048GL19248</td>
<td>248MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Adv</td>
<td>EFR32MG12P432F1024</td>
<td>40MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>anced IoT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Silicon Labs EFM32 development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

Stable

```plaintext
; Latest stable version
[env:latest_stable]
platform = siliconlabsefm32
board = ...

; Custom stable version
[env:custom_stable]
platform = siliconlabsefm32@x.y.z
board = ...
```
Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-siliconlabsefm32.git
board = ...
```
## Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-silabs</td>
<td>SiliconLabs HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinybor</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a minimal set of standard cryptography primitives for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARMv8-M and Zephyr framework</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>
Warning: Linux Users:
- Install “udev” rules 99-platformio-udev.rules
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td><strong>Zephyr RTOS</strong></td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

Silicon Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EFM32GG-STK3700 Giant Gecko</strong></td>
<td>On-board</td>
<td>EFM32GG990F1024</td>
<td>48MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>EFM32LG-STK3600 Leopard Gecko</strong></td>
<td>On-board</td>
<td>EFM32LG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td><strong>EFM32WG-STK3800 Wonder Gecko</strong></td>
<td>On-board</td>
<td>EFM32WG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td><strong>EFM32ZG-STK3200 Zero Gecko</strong></td>
<td>On-board</td>
<td>EFM32ZG222F32</td>
<td>24MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td><strong>SLSTK3400A USB-enabled Happy Gecko</strong></td>
<td>On-board</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td><strong>SLSTK3401A Pearl Gecko PG1</strong></td>
<td>On-board</td>
<td>EFM32PG1B200F256GM48</td>
<td>40MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td><strong>SLSTK3701A Giant Gecko S1</strong></td>
<td>On-board</td>
<td>EFM32GG11B820F2048GL192</td>
<td>48MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td><strong>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</strong></td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
<td>40MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
ST STM32

Configuration \( \text{platform} = \text{ststm32} \)

The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

For more detailed information please visit vendor site.

Contents

- Tutorials
- Configuration
- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Tutorials

- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

Configuration

Switching between Arduino cores

There are three different Arduino cores for STM32 microcontrollers: STM32Duino, Arduino STM32 (maple) and STM32L0. All of them have been developed independently, therefore, have different functionality and set of internal libraries. By default, official STM32Duino core is used (except cases when a board supports only one specific core). Some of the boards support all three cores. To change the core you can use `board_build.core` option that needs to be added to `build_flags`:

An example of “platformio.ini” (Project Configuration File) with maple core

```
[env:hy_tinystm103tb]
platform = ststm32
framework = arduino
board = hy_tinystm103tb
board_build.core = maple
```

STM32Duino configuration system

STM32Duino core has several options that can be configured using the next configuration flags in `build_flags` section of “platformio.ini” (Project Configuration File):
### PlatformIO Documentation, Release 5.0.5a1

**Table 12: C/C++ standard library configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_STANDARD_LIB</td>
<td>Disable Newlib Nano library</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_NANOLIB_FLOAT_PRINTF</td>
<td>Newlib Nano + float printf support</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_NANOLIB_FLOAT_SCANF</td>
<td>Newlib Nano + float scanf support</td>
</tr>
</tbody>
</table>

**Table 13: USART Configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_SERIAL_WITHOUT_GENERIC</td>
<td>Enabled (no generic Serial)</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_SERIAL_DISABLED</td>
<td>Disabled (no Serial support)</td>
</tr>
</tbody>
</table>

**Table 14: USB Configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_ENABLE_CDC</td>
<td>CDC (generic Serial supersed U(S)ART)</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_ENABLE_CDC_WITHOUT_SERIAL</td>
<td>CDC (no generic Serial)</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_ENABLE_HID</td>
<td>HID (keyboard and mouse)</td>
</tr>
</tbody>
</table>

**Table 15: USB Speed Configuration**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_USB_HIGHSPEED</td>
<td>High Speed mode</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_USB_HIGHSPEED_FULLMODE</td>
<td>High Speed in Full Speed mode</td>
</tr>
</tbody>
</table>

Example:

```ini
[env:nucleo_f401re]
platform = ststm32
framework = arduino
board = nucleo_f401re
build_flags =
-D PIO_FRAMEWORK_ARDUINO_ENABLE_CDC
-D PIO_FRAMEWORK_ARDUINO_NANOLIB_FLOAT_PRINTF
-D PIO_FRAMEWORK_ARDUINO_USB_HIGHSPEED_FULLMODE
```

**Maple STM32 configuration system**

In this core the USB peripheral (STM32F4 boards only) can be configured using the next configuration flags in `build_flags` section of "platformio.ini" (Project Configuration File):

**Table 16: USB Configuration for STM32F4 boards**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_USB_SERIAL</td>
<td>USB serial (CDC)</td>
</tr>
<tr>
<td>ENABLE_USB_MASS_STORAGE</td>
<td>USB Mass Storage (MSC)</td>
</tr>
</tbody>
</table>

Example:
Arduino STM32L0 configuration system

Arduino STM32L0 core has several options that can be configured using the next configuration flags in `build_flags` section of "platformio.ini" (Project Configuration File):

Table 17: USB Configuration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_ENABLE_MASS_STORAGE</td>
<td>Serial + Mass Storage</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_ENABLE_HID</td>
<td>Serial + Keyboard + Mouse</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_ENABLE_MASS_STORAGE_HID</td>
<td>Serial + Mass Storage + Keyboard + Mouse</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_NO_USB</td>
<td>No USB</td>
</tr>
</tbody>
</table>

Table 18: FS Configuration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_FS_SDCARD</td>
<td>SDCARD (SPI)</td>
</tr>
<tr>
<td>PIO_FRAMEWORK_ARDUINO_FS_SFLASH</td>
<td>SFLASH (SPI)</td>
</tr>
</tbody>
</table>

Example:

```
[env:cricket_l082cz]
platform = ststm32
framework = arduino
board = cricket_l082cz
build_flags = 
-D PIO_FRAMEWORK_ARDUINO_ENABLE_MASS_STORAGE
```

Examples

Examples are listed from ST STM32 development platform repository:

- mbed-rtos-wifi
- zephyr-blink
- stm32cube-ll-blink
- libopencm3-blink
- stm32cube-hal-usb-host-hid
- libopencm3-usb-cdcacm
- cmsis-blink
- stm32cube-hal-iap
- arduino-blink
- `stm32cube-hal-extmem-boot`
- `spl-blink`
- `stm32cube-hal-lcd`
- `mbed-rtos-mesh-minimal`
- `arduino-mxchip-sensors`
- `zephyr-cpp-synchronization`
- `zephyr-net-https-client`
- `mbed-rtos-blink-baremetal`
- `mbed-rtos-filesystem`
- `stm32cube-hal-wifi-client`
- `mbed-rtos-usb-keyboard`
- `mbed-rtos-ethereum-tls`
- `stm32cube-hal-blink`
- `zephyr-drivers-can`
- `arduino-mxchip-azureiot`
- `arduino-mxchip-filesystem`
- `stm32cube-hal-usb-device-dfu`
- `stm32cube-hal-blink-clangd`
- `arduino-internal-libs`
- `arduino-external-libs`
- `arduino-mxchip-wifiscan`
- `mbed-rtos-events`
- `mbed-legacy-examples`
- `libopencm3-1bitsy`
- `zephyr-subsys-usb-hid-mouse`
- `mbed-rtos-serial`
- `mbed-rtos-custom-target`
- `stm32cube-hal-eprom-emulation`

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - **On-Board Debug Tools**
  - **External Debug Tools**
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

### On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>LA76DMW1K</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>STM32WB55RG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192.00KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F33488DISCOVERY</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F4291DISCOVERY</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F4691DISCOVERY</td>
<td>STM32F469NHIY6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F474GDISCOVERY</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>STM32F769NHIY6</td>
<td>216MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070R8</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Name</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------</td>
<td>-----------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>STM32F410RB6</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F740ZG</td>
<td>STM32F740ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>STM32F756ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H745ZI-Q</td>
<td>STM32H745ZIT6</td>
<td>480MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>STM32L073RZT6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>STM32L412KBT6</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>STM32L432KCUT6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>STM32L433RCT6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>STM32F0308T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F051R8DISCOVERY</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F030VCDiscovery</td>
<td>STM32F030VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F0407VDiscovery</td>
<td>STM32F0407GVT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>STM32L073ZVT6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>STM32L152RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDiscovery</td>
<td>STM32L152RBT6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>STM32L476G</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST32F5708-DK</td>
<td>STM32F5708N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</td>
</tr>
<tr>
<td>ST32H747I-DISCO</td>
<td>STM32H747IHX6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>STM32F439V1</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Continued on next page
## Table 19 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sakura.io Evaluation Board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><em>u</em>-blox C030-R410M IoT</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td><em>u</em>-blox ODIN-W2</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with **Debugging** but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Odinkey</td>
<td>STM32F427VTH6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Cicada-L082CZ</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Econode-L082CZ</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Electrosmith Daisy</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>F407VG</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FF407M1</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>192.00KB</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 20 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple</td>
<td>STM32F103RB6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>N2+</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>NAMote72</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>STM32G071RB</td>
<td>64MHz</td>
<td>128KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>STM32G431KB</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>STM32G431RB</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>STM32G474RE</td>
<td>170MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>PYBStick26 Duino</td>
<td>STM32F407RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBStick 26 Pro</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Piconomix PX-HER0</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>PrntrBoard V2</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32J0C-EVAL</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32J373C-EVAL</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F072-VAL</td>
<td>STM32F072BVT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM. 16k Flash)</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM. 32k Flash)</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM. 64k Flash)</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM. 128k Flash)</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM. 16k Flash)</td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM. 32k Flash)</td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM. 64k Flash)</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM. 128k Flash)</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM. 256k Flash)</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM. 384k Flash)</td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM. 512k Flash)</td>
<td>STM32F103RE</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM. 768k Flash)</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM. 1024k Flash)</td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103T4 (6k RAM. 16k Flash)</td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM. 32k Flash)</td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM. 64k Flash)</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103V8 (20k RAM, 64k Flash)</td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>STM32F103VB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VCT6 (48k RAM, 256k Flash)</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VDT6 (64k RAM, 384k Flash)</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VET6 (96k RAM, 512k Flash)</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (96k RAM, 768k Flash)</td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM, 1024k Flash)</td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128k Flash)</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM, 128k Flash)</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM, 256k Flash)</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM, 384k Flash)</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512k Flash)</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM, 128k Flash)</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (96k RAM, 256k Flash)</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 384k Flash)</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410R8 (32k RAM, 64k Flash)</td>
<td>STM32F410R8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM, 256k Flash)</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM, 512k Flash)</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM, 1024k Flash)</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1536k Flash)</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (256k RAM, 1024k Flash)</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1536k Flash)</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F405RGT6 (192k RAM, 512k Flash)</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 20 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorTile.box</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>VAkE v1.0</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of ST STM32 development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

Stable

```ini
; Latest stable version
[env:latest_stable]
platform = ststm32
board = ...

; Custom stable version
[env:custom_stable]
platform = ststm32@x.y.z
board = ...
```

Upstream

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-ststm32.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinostm32mxchip</td>
<td>Arduino Wiring-based Framework for the Azure MXChip IoT DevKit</td>
</tr>
<tr>
<td>framework-arduinoststm32</td>
<td>Arduino Wiring-based Framework for ST STM32 microcontrollers</td>
</tr>
<tr>
<td>framework-arduinoststm32-maple</td>
<td>Arduino Wiring-based Framework for ST STM32 microcontrollers (Maple Core)</td>
</tr>
<tr>
<td>framework-arduinoststm32l0</td>
<td>Arduino Wiring-based Framework for ST STM32 microcontrollers (ST STM32L0 Core)</td>
</tr>
<tr>
<td>framework-cmsis</td>
<td>Vendor-independent hardware abstraction layer for the Cortex-M processor series</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>framework-zephyr</td>
<td>Zephyr is a new generation, scalable, optimized, secure RTOS for multiple</td>
</tr>
<tr>
<td></td>
<td>hardware architectures</td>
</tr>
<tr>
<td>framework-zephyr-canopennode</td>
<td>canopennode Zephyr module</td>
</tr>
<tr>
<td>framework-zephyr-civetweb</td>
<td>Zephyr module for CivetWeb Embedded C/C++ web server</td>
</tr>
<tr>
<td>framework-zephyr-cmsis</td>
<td>Software Interface Standard for Arm Cortex-based Microcontrollers and Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-fats</td>
<td>Zephyr module for FATFS filesystem</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-hal-stm32</td>
<td>ST STM32 HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinyborz</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinyborz</td>
<td>The TinyBorz Library provides an implementation for constrained devices of a</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>minimal set</td>
</tr>
<tr>
<td>framework-zephyr-fatfs</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-hal-st</td>
<td>Zephyr module for the official libraries provided by STMicroelectronics</td>
</tr>
<tr>
<td>framework-zephyr-hal-stm32</td>
<td>ST STM32 HAL for Zephyr framework</td>
</tr>
<tr>
<td>framework-zephyr-libmetal</td>
<td>Zephyr module for HAL abstraction layer used by open-amp</td>
</tr>
<tr>
<td>framework-zephyr-littlefs</td>
<td>Zephyr module for littlefs filesystem</td>
</tr>
<tr>
<td>framework-zephyr-loramac-node</td>
<td>Zephyr module for LoRaWAN endpoint stack implementation</td>
</tr>
<tr>
<td>framework-zephyr-lvgl</td>
<td>Zephyr module for LittlevGL - an Open-source Embedded GUI Library</td>
</tr>
<tr>
<td>framework-zephyr-mbedtls</td>
<td>mbedTLS module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-mcuboot</td>
<td>Zephyr module for MCUboot - a secure bootloader for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mcumgr</td>
<td>Zephyr module for mcumgr management library for 32-bit MCUs</td>
</tr>
<tr>
<td>framework-zephyr-mipi-sys-t</td>
<td>Zephyr module for MIPI System Software Trace</td>
</tr>
<tr>
<td>framework-zephyr-open-amp</td>
<td>Zephyr module for Open Asymmetric Multi Processing (OpenAMP) framework</td>
</tr>
<tr>
<td>framework-zephyr-openthread</td>
<td>OpenThread module for Zephyr</td>
</tr>
<tr>
<td>framework-zephyr-segger</td>
<td>Zephyr module for Segger RTT</td>
</tr>
<tr>
<td>framework-zephyr-tinyborz</td>
<td>Zephyr module for Concise Binary Object Representation Library</td>
</tr>
<tr>
<td>framework-zephyr-tinycrypt</td>
<td>The TinyCrypt Library provides an implementation for constrained devices of a</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>framework-zephyr-trusted-firmware-m</td>
<td>Trusted Firmware M provides a reference implementation of secure world software for ARMv8-M and Zephyr framework</td>
</tr>
<tr>
<td>tool-cmake</td>
<td>CMake is an open-source, cross-platform family of tools designed to build, test and package software</td>
</tr>
<tr>
<td>tool-dfuutil</td>
<td>Device Firmware Upgrade Utilities</td>
</tr>
<tr>
<td>tool-dtc</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-gperf</td>
<td>GNU gperf is a perfect hash function generator</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Device tree compiler</td>
</tr>
<tr>
<td>tool-ldscripts-ststm32</td>
<td>Linker scripts pack for STMicroelectronics STM32 platform</td>
</tr>
<tr>
<td>tool-ninja</td>
<td>Ninja is a small build system with a focus on speed</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and Testing</td>
</tr>
<tr>
<td>tool-stm32duino</td>
<td>STM32Duino Tools</td>
</tr>
<tr>
<td>toolchain-gccarmnoneeabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

**Warning:** Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article [Enable serial port on Raspberry Pi](#)

**Windows Users:**
Please check that you have a correctly installed USB driver from board manufacturer
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CM-SIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32C</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### Boards

**Note:**

- You can list pre-configured boards by *pio boards* command or PlatformIO Boards Explorer.
- For more detailed board information please scroll the tables below by horizontally.

### 1BitSquared

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
## 96Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## AfroFlight

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## Airbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wraith V1 ESC</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
</tbody>
</table>

## Armed

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## Armstrap

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrap Eagle 1024</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>On-board</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 312</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## Avnet Silica

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Sensor Node</td>
<td>On-board</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### Diymore

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F407VG</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Econode

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Econode-L082CZ</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Electrosmith

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrosmith Daisy</td>
<td>External</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Espotel

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espotel LoRa Module</td>
<td>External</td>
<td>STM32F441RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### FYSETC

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYSETC S6</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F103C8</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM. 16k Flash)</td>
<td>External</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM. 32k Flash)</td>
<td>External</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM. 64k Flash)</td>
<td>External</td>
<td>STM32F103C8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM. 128k Flash)</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM. 16k Flash)</td>
<td>External</td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM. 32k Flash)</td>
<td>External</td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>External</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM, 384k Flash)</td>
<td>External</td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM, 768k Flash)</td>
<td>External</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103T4 (6k RAM, 16k Flash)</td>
<td>External</td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM, 32k Flash)</td>
<td>External</td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103T8V8 (20k RAM, 64k Flash)</td>
<td>External</td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>External</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>External</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (96k RAM, 768k Flash)</td>
<td>External</td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384k Flash)</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM, 768k Flash)</td>
<td>External</td>
<td>STM32F103ZF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F103ZG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128k Flash)</td>
<td>External</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM, 128k Flash)</td>
<td>External</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM, 384k Flash)</td>
<td>External</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM, 128k Flash)</td>
<td>External</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 384k Flash)</td>
<td>External</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM, 64k Flash)</td>
<td>External</td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>External</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM, 64k Flash)</td>
<td>External</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
Table 22 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F413RG (320k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1536k Flash)</td>
<td>External</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>External</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>External</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 1536k Flash)</td>
<td>External</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM, 256k Flash)</td>
<td>External</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>External</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

HY

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny STM103T</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

LeafLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
</tbody>
</table>

MXChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Malyan

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M200 V2</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
</tbody>
</table>
### Midatronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKR Sharky</td>
<td>External</td>
<td>STM32WB55CG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192.00KB</td>
</tr>
</tbody>
</table>

### MultiTech

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTS Dragonfly</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Netduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2+</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olimexino-STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### PYBStick

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYBStick26 Duino</td>
<td>External</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBStick 26 Pro</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Piconomix

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piconomix PX-HERO</td>
<td>External</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>
PrntrBoard

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrntrBoard V2</td>
<td>External</td>
<td>STM32F-407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

RAK

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

RUMBA

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer control board</td>
<td>External</td>
<td>STM32F-446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

RemRam

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer controller</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

RobotDyn

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F303CC</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

RushUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

ST

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>On-board</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>External</td>
<td>STM32F407VE16</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black STM32F407VG</td>
<td>External</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>External</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>External</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>External</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>External</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>External</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>External</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>External</td>
<td>84MHz</td>
<td>128KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>External</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>External</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>External</td>
<td>170MHz</td>
<td>128KB</td>
<td>512KB</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>On-board</td>
<td>64MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>External</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>On-board</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>On-board</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>On-board</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>On-board</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F4291DISCOVERY</td>
<td>On-board</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F4691DISCOVERY</td>
<td>On-board</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>On-board</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>On-board</td>
<td>216MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>On-board</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>On-board</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>On-board</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>On-board</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>On-board</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>On-board</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>On-board</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>On-board</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>On-board</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>On-board</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>On-board</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>On-board</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>On-board</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>On-board</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>On-board</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>On-board</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>On-board</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>On-board</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>On-board</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>On-board</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>On-board</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>On-board</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>On-board</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>On-board</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>On-board</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>On-board</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>On-board</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>On-board</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F446ZE</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>On-board</td>
<td>STM32F746ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>On-board</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767Z1I</td>
<td>On-board</td>
<td>STM32F767ZI</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H743Z1I</td>
<td>On-board</td>
<td>STM32H743ZI</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H745Z1-Q</td>
<td>On-board</td>
<td>STM32H745ZI</td>
<td>480MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>On-board</td>
<td>STM32L011K4</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>On-board</td>
<td>STM32L031K6</td>
<td>32MHz</td>
<td>16KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>On-board</td>
<td>STM32L053R8</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>On-board</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>On-board</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>On-board</td>
<td>STM32L412KBU6</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>On-board</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>On-board</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>On-board</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>On-board</td>
<td>STM32L476RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>On-board</td>
<td>STM32L486RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>On-board</td>
<td>STM32L496ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>On-board</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST STM32F030DISCOVERY</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>On-board</td>
<td>STM32F407VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>On-board</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>On-board</td>
<td>STM32L073VZT6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>On-board</td>
<td>STM32L152RB6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>On-board</td>
<td>STM32F100RBT6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>External</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>External</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>External</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</td>
</tr>
<tr>
<td>STM32H747T1-DISCO</td>
<td>On-board</td>
<td>STM32H747THX6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

**SeeedStudio**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Arch Max</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>On-board</td>
<td>STM32F439V1</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

**Semtech**

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMote72</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
## TauLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparky V1 F303</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

## ThunderPack

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThunderPack v1.0</td>
<td>External</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## Tlera Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cicada-L082CZ</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## VAE

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAE v1.0</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## VCCGND

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Waveshare

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveshare Open103Z</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## WeAct

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeAct Blackpill V2.0 (Blackpill F411CE)</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### rhomb.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L476DMW1K</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### sakura.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sakura.io Evaluation Board</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed Connect Cloud</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>On-board</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### ST STM8

**Configuration** `platform = ststm8`

The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.

For more detailed information please visit vendor site.

### Contents

- **Examples**
- **Debugging**
- **Stable and upstream versions**
- **Packages**
- **Frameworks**
- **Boards**

### Examples

Examples are listed from ST STM8 development platform repository:

- spl-blink
- arduino-fade-all-pins
• spl-uart
• spl-flash
• arduino-internal-lib
• arduino-ping-hc04

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST STM8S-DISCOVERY</td>
<td>STM8S105C6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of ST STM8 development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

Stable

```ini
; Latest stable version
[env:latest_stable]
platform = ststm8
board = ...

; Custom stable version
[env:custom_stable]
```

(continues on next page)
platform = ststm8@x.y.z
board = ...

Upstream

[env:upstream_develop]
platform = https://github.com/platformio/platform-ststm8.git
board = ...

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinoststm8</td>
<td>An Arduino-like programming API for the STM8 microcontrollers</td>
</tr>
<tr>
<td>framework-ststm8spl</td>
<td>Standard peripheral library for ST STM8S/A microcontrollers</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing</td>
</tr>
<tr>
<td>tool-stm8binutils</td>
<td>STM8 toolchain with GDB debugger</td>
</tr>
<tr>
<td>tool-stm8tools</td>
<td>Upload tools for ST STM8 microcontrollers</td>
</tr>
<tr>
<td>toolchain-sdcc</td>
<td>Small Device C compiler suite</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules 99-platformio-udev.rules
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
- Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
</tbody>
</table>

Boards

Note:
• You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer

• For more detailed board information please scroll the tables below by horizontally.

### ST

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST STM8S-DISCOVERY</td>
<td>On-board</td>
<td>STM8S105C6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST STM8S103F3 Breakout Board</td>
<td>No</td>
<td>STM8S103F3P6</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ST STM8S105K4T6 Breakout Board</td>
<td>No</td>
<td>STM8S105K4T6</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### sduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sduino MB (STM8S208MBT6B)</td>
<td>No</td>
<td>STM8S208MBT6</td>
<td>16MHz</td>
<td>128KB</td>
<td>6KB</td>
</tr>
<tr>
<td>sduino UNO (STM8S105K6)</td>
<td>No</td>
<td>STM8S105K6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Teensy

**Configuration** `platform = teensy`

Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

For more detailed information please visit [vendor site](#).

### Contents

- Configuration
- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

### Configuration

- Optimization
- USB Features
Optimization

(valid only for Teensy LC, Teensy 3.0-3.6)

You can control firmware optimization via special macro/define using `build_flags` in “platformio.ini” (Project Configuration File):

- `-D TEENSY_OPT_FASTER`, default
- `-D TEENSY_OPT_FASTER_LTO`
- `-D TEENSY_OPT_FAST`
- `-D TEENSY_OPT_FAST_LTO`
- `-D TEENSY_OPT_FASTEST`
- `-D TEENSY_OPT_FASTEST_LTO`
- `-D TEENSY_OPT_FASTEST_PURE_CODE`, valid only for Teensy 3.5-3.6
- `-D TEENSY_OPT_FASTEST_PURE_CODE_LTO`, valid only for Teensy 3.5-3.6
- `-D TEENSY_OPT_DEBUG`
- `-D TEENSY_OPT_DEBUG_LTO`
- `-D TEENSY_OPT_SMALLEST_CODE`
- `-D TEENSY_OPT_SMALLEST_CODE_LTO`

The only one macro can be used in per one build environment. Also, you can see verbose build using `-v`, `--verbose` option for `pio run` command.

Example:

Let’s set optimization for the smallest code

```
[env:teensy_hid_device]
platform = teensy
framework = arduino
board = teensy36
build_flags = -D TEENSY_OPT_SMALLEST_CODE
```

USB Features

If you want to use Teensy USB Features, you need to add special macro/define using `build_flags`:

- `-D USB_SERIAL`
- `-D USB_KEYBOARDOONLY`
- `-D USB_TOUCHSCREEN`
- `-D USB_HID_TOUCHSCREEN`
- `-D USB_HID`
- `-D USB_SERIAL_HID`
- `-D USB_MIDI`
- `-D USB_MIDI4`
- `-D USB_MIDI16`
• `-D USB_MIDI_SERIAL`
• `-D USB_MIDI4_SERIAL`
• `-D USB_MIDI16_SERIAL`
• `-D USB_AUDIO`
• `-D USB_MIDI_AUDIO_SERIAL`
• `-D USB_MIDI16_AUDIO_SERIAL`
• `-D USB_MTDISK`
• `-D USB_RAWHID`
• `-D USB_FLIGHTSIM`
• `-D USB_FLIGHTSIM_JOYSTICK`
• `-D USB_EVERYTHING`
• `-D USB_DISABLED`

A default macro is set to `-D USB_SERIAL` if no one is specified.

Example:

```
[env:teensy_hid_device]
platform = teensy
framework = arduino
board = teensy20
build_flags = -D USB_RAWHID
```

See Teensy USB Examples.

**Examples**

Examples are listed from Teensy development platform repository:

• mbed-blink
• mbed-serial
• mbed-events
• arduino-blink
• arduino-hid-usb-mouse
• arduino-internal-libs
• mbed-dsp

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - **External Debug Tools**
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

### External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>MK64FX512</td>
<td>120MHz</td>
<td>512KB</td>
<td>255.99KB</td>
</tr>
<tr>
<td>Teensy 3.6</td>
<td>MK66FX1M0</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Teensy 4.0</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>1.94MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy 4.1</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>7.75MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>MKL26Z64</td>
<td>48MHz</td>
<td>62KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### Stable and upstream versions

You can switch between stable releases of Teensy development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

#### Stable

```ini
; Latest stable version
[env:latest_stable]
platform = teensy
board = ...

; Custom stable version
[env:custom_stable]
platform = teensy@x.y.z
board = ...
```

#### Upstream

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-teensy.git
board = ...
```
Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinoteensy</td>
<td>Arduino Wiring-based Framework for Teensy boards</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-teensy</td>
<td>Upload tools for Teensy boards</td>
</tr>
<tr>
<td>toolchain-atmelavr</td>
<td>GCC Toolchain for Microchip AVR microcontrollers</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

Warning: Linux Users:

- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:

Teensy programming uses only Windows built-in HID drivers. When Teensy is programmed to act as a USB Serial device, Windows XP, Vista, 7 and 8 require this serial driver is needed to access the COM port your program uses. No special driver installation is necessary on Windows 10.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
Teensy

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teensy 2.0</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Teensy 3.0</td>
<td>No</td>
<td>MK20DX128</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>External</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>External</td>
<td>MK64FX512</td>
<td>120MHz</td>
<td>512KB</td>
<td>255.99KB</td>
</tr>
<tr>
<td>Teensy 3.6</td>
<td>External</td>
<td>MK66FX1M0</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Teensy 4.0</td>
<td>External</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>1.94MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy 4.1</td>
<td>External</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>7.75MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>External</td>
<td>MKL26Z64</td>
<td>48MHz</td>
<td>62KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Teensy++ 2.0</td>
<td>No</td>
<td>AT90USB1286</td>
<td>16MHz</td>
<td>127KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

**TI MSP430**

**Configuration** platform = timsp430

MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.

For more detailed information please visit vendor site.

**Examples**

Examples are listed from TI MSP430 development platform repository:

- arduino-blink
- native-blink
- arduino-internal-libs

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

### On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI LaunchPad MSP-EXP430FR5739LP</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>15.37KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>MSP430F5529</td>
<td>25MHz</td>
<td>47KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2311LP</td>
<td>MSP430FR2311</td>
<td>16MHz</td>
<td>3.75KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2433LP</td>
<td>MSP430FR2433</td>
<td>8MHz</td>
<td>15KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR4133LP</td>
<td>MSP430FR4133</td>
<td>8MHz</td>
<td>15KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5969LP</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5994LP</td>
<td>MSP430FR5994</td>
<td>16MHz</td>
<td>256KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</td>
<td>MSP430G2231</td>
<td>1MHz</td>
<td>2KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</td>
<td>MSP430G2452</td>
<td>16MHz</td>
<td>8KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2553LP</td>
<td>MSP430G2553</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
</tbody>
</table>

### Stable and upstream versions

You can switch between stable releases of TI MSP430 development platform and the latest upstream version using platform option in “platformio.ini” (Project Configuration File) as described below.

#### Stable

```ini
; Latest stable version
[env:latest_stable]
platform = timsp430
board = ...

; Custom stable version
[env:custom_stable]
platform = timsp430@x.y.z
board = ...
```

1.10. Development Platforms
Upstream

```yaml
[env:upstream_develop]
platform = https://github.com/platformio/platform-timsp430.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-energiamsp430</td>
<td>Energia Wiring-based Framework for Texas Instruments MSP430 microcontrollers</td>
</tr>
<tr>
<td>tool-dslite</td>
<td>Texas Instruments DSLite (UniFlash)</td>
</tr>
<tr>
<td>tool-mspdebug</td>
<td>MSPDebug is a free debugger for use with MSP430 MCUs</td>
</tr>
<tr>
<td>toolchain-timsp430</td>
<td>A port of the GNU C Compiler (GCC) and GNU Binutils (as, ld) for the embedded processor MSP430</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
- Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
## TI

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI FraunhPAd MSP-EXP430FR5739LP</td>
<td>On-board</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>15.37KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>On-board</td>
<td>MSP430F5529</td>
<td>25MHz</td>
<td>47KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2311LP</td>
<td>On-board</td>
<td>MSP430FR2311</td>
<td>16MHz</td>
<td>3.75KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2433LP</td>
<td>On-board</td>
<td>MSP430FR2433</td>
<td>8MHz</td>
<td>15KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR4133LP</td>
<td>On-board</td>
<td>MSP430FR4133</td>
<td>8MHz</td>
<td>15KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5969LP</td>
<td>On-board</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5994LP</td>
<td>On-board</td>
<td>MSP430FR5994</td>
<td>16MHz</td>
<td>256KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</td>
<td>On-board</td>
<td>MSP430G2231</td>
<td>1MHz</td>
<td>2KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</td>
<td>On-board</td>
<td>MSP430G2452</td>
<td>16MHz</td>
<td>8KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2553LP</td>
<td>On-board</td>
<td>MSP430G2553</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
</tbody>
</table>

### TI TIVA

**Configuration** \( \text{platform} = \text{titiva} \)

Texas Instruments TM4C12x MCUs offer the industry's most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.

For more detailed information please visit [vendor site](#).

## Contents

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards
Examples

Examples are listed from TI TIVA development platform repository:

- libopencm3-blink
- libopencm3-usb-cdcacm
- arduino-blink
- native-blink
- arduino-internal-libs

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - **On-Board Debug Tools**

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>LPTM4C1294NCMDT</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of TI TIVA development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.
Stable

```plaintext
; Latest stable version
[env:latest_stable]
platform = titiva
board = ...

; Custom stable version
[env:custom_stable]
platform = titiva@x.y.z
board = ...
```

Upstream

```plaintext
[env:upstream_develop]
platform = https://github.com/platformio/platform-titiva.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-energiativa</td>
<td>Energia Wiring-based Framework for Texas Instruments TIVA C Series</td>
</tr>
<tr>
<td>framework-libopencm3</td>
<td>Open source ARM Cortex-M microcontroller library</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>Open On-Chip Debugger. Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
- Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>On-board</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>On-board</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>On-board</td>
<td>LPTM4C1294NCPT</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

WIZNet W7500

Configuration `platform = wiznet7500`

The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things. For more detailed information please visit vendor site.

Contents

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Examples

Examples are listed from WIZNet W7500 development platform repository:
- mbed-blink
- mbed-serial
- mbed-rtos
- mbed-events
• mbed-dsp

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - **On-Board Debug Tools**

**Tools & Debug Probes**

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (**Project Configuration File**).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

**On-Board Debug Tools**

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZwiki-W7500</td>
<td>WIZNET7500</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>WIZNET7500ECO</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>WIZNET7500P</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
</tbody>
</table>

**Stable and upstream versions**

You can switch between stable releases of WIZNet W7500 development platform and the latest upstream version using **platform** option in “platformio.ini” (**Project Configuration File**) as described below.

**Stable**

```
; Latest stable version
[env:latest_stable]
platform = wiznet7500
board = ...

; Custom stable version
[env:custom_stable]
platform = wiznet7500@x.y.z
board = ...
```
Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-wiznet7500.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-mbed</td>
<td>Arm Mbed OS is a platform operating system designed for the internet of things</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>Software and Documentation Pack for SEGGER J-Link debug probes</td>
</tr>
<tr>
<td>tool-pyocd</td>
<td>Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>GNU toolchain for Arm Cortex-M and Cortex-R processors</td>
</tr>
</tbody>
</table>

Warning: Linux Users:
- Install “udev” rules `99-platformio-udev.rules`
- Raspberry Pi users, please read this article Enable serial port on Raspberry Pi.

Windows Users:
- Please check that you have a correctly installed USB driver from board manufacturer

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
WIZNet

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZwiki-W7500</td>
<td>On-board</td>
<td>WIZNET7500</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>On-board</td>
<td>WIZNET7500ECO</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>On-board</td>
<td>WIZNET7500P</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
</tbody>
</table>

1.10.2 Desktop

Native

**Configuration platform** = native

Native development platform is intended to be used for desktop OS. This platform uses built-in toolchains (preferable based on GCC), frameworks, libs from particular OS where it will be run.

For more detailed information please visit vendor site.

### Contents

- Examples
- Stable and upstream versions

Examples

Examples are listed from Native development platform repository:

- hello-world

Stable and upstream versions

You can switch between stable releases of Native development platform and the latest upstream version using `platform` option in "platformio.ini" (Project Configuration File) as described below.

**Stable**

```ini
; Latest stable version
[env:latest_stable]
platform = native
board = ...

; Custom stable version
[env:custom_stable]
platform = native@x.y.z
board = ...
```

1.10. Development Platforms
Linux ARM

Configuration \textit{platform} = \texttt{linux\_arm}

Linux ARM is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS X, Linux ARM) you can build native application for Linux ARM platform.

For more detailed information please visit vendor site.

Contents

- Examples
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Examples

Examples are listed from Linux ARM development platform repository:

- wiringpi-serial
- wiringpi-blink

Stable and upstream versions

You can switch between stable releases of Linux ARM development platform and the latest upstream version using \textit{platform} option in \textit{platformio.ini} (Project Configuration File) as described below.

Stable

\begin{verbatim}
; Latest stable version
[env:latest_stable]
platform = linux_arm
board = ...

; Custom stable version
[env:custom_stable]
platform = linux_arm@x.y.z
board = ...
\end{verbatim}
Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-linux_arm.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-</td>
<td>WiringPi is a PIN based GPIO access library written in C for the BCM2835, BCM2836 and BCM2837 SoC devices used in all Raspberry Pi</td>
</tr>
<tr>
<td>wiringpi</td>
<td></td>
</tr>
<tr>
<td>toolchain-</td>
<td>GCC Toolchain for Linux ARM GNU EABI</td>
</tr>
<tr>
<td>gccarmlinuxgnueabi</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiringPi</td>
<td>WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry Pi. It’s designed to be familiar to people who have used the Arduino ‘wiring’ system</td>
</tr>
</tbody>
</table>

Boards

**Note:**

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

Raspberry Pi

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi 1 Model B</td>
<td>No</td>
<td>BCM2835</td>
<td>700MHz</td>
<td>512MB</td>
<td>512MB</td>
</tr>
<tr>
<td>Raspberry Pi 2 Model B</td>
<td>No</td>
<td>BCM2836</td>
<td>900MHz</td>
<td>1GB</td>
<td>1GB</td>
</tr>
<tr>
<td>Raspberry Pi 3 Model B</td>
<td>No</td>
<td>BCM2837</td>
<td>1200MHz</td>
<td>1GB</td>
<td>1GB</td>
</tr>
<tr>
<td>Raspberry Pi Zero</td>
<td>No</td>
<td>BCM2835</td>
<td>1000MHz</td>
<td>512MB</td>
<td>512MB</td>
</tr>
</tbody>
</table>

Linux i686

**Configuration** `platform = linux_i686`

Linux i686 (32-bit) is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS X or Linux 32-bit) you can build native application for Linux i686 platform.

For more detailed information please visit vendor site.

1.10. Development Platforms
Examples

Examples are listed from Linux i686 development platform repository:

- hello-world

Stable and upstream versions

You can switch between stable releases of Linux i686 development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

```
[env:latest_stable]
platform = linux_i686
board = ...

; Custom stable version
[env:custom_stable]
platform = linux_i686@x.y.z
board = ...
```

**Upstream**

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-linux_i686.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolchain-gcc-linux32</td>
<td>GCC Toolchain for Linux i686 machines</td>
</tr>
</tbody>
</table>

**Linux x86_64**

Configuration `platform = linux_x86_64`
Linux x86_64 (64-bit) is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS X or Linux 64-bit) you can build native application for Linux x86_64 platform.

For more detailed information please visit vendor site.

---

**Contents**

- **Examples**
- **Stable and upstream versions**
- **Packages**

---

### Examples

Examples are listed from Linux x86_64 development platform repository:

- hello-world

### Stable and upstream versions

You can switch between stable releases of Linux x86_64 development platform and the latest upstream version using `platform` option in "platformio.ini" (Project Configuration File) as described below.

#### Stable

```ini
; Latest stable version
[env:latest_stable]
platform = linux_x86_64
board = ...

; Custom stable version
[env:custom_stable]
platform = linux_x86_64@x.y.z
board = ...
```

#### Upstream

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-linux_x86_64.git
board = ...
```

### Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolchain-gcc64</td>
<td>GCC Toolchain for Linux x86_64 machines</td>
</tr>
</tbody>
</table>
Windows x86

Configuration `platform = windows_x86`

Windows x86 (32-bit) is a metafamily of graphical operating systems developed and marketed by Microsoft. Using host OS (Windows, Linux 32/64 or Mac OS X) you can build native application for Windows x86 platform.

For more detailed information please visit vendor site.

Contents

- Examples
- Stable and upstream versions
- Packages

Examples

Examples are listed from Windows x86 development platform repository:

- hello-world

Stable and upstream versions

You can switch between stable releases of Windows x86 development platform and the latest upstream version using `platform` option in “platformio.ini” (Project Configuration File) as described below.

**Stable**

```
; Latest stable version
[env:latest_stable]
platform = windows_x86
board = ...

; Custom stable version
[env:custom_stable]
platform = windows_x86@x.y.z
board = ...
```

**Upstream**

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-windows_x86.git
board = ...
```
Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>toolchain-gccmingw32</td>
<td>MinGW - Minimalist GNU for Windows</td>
</tr>
</tbody>
</table>

1.11 Frameworks

1.11.1 Arduino

**Configuration**  \( framework = \text{arduino} \)

Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

For more detailed information please visit vendor site.

Contents

- Tutorials
- Configuration
- Debugging
- Examples
- Platforms
- Boards

Tutorials

- Get started with Arduino and ESP32-DevKitC: debugging and unit testing
- Arduino and Nordic nRF52-DK: debugging and unit testing

Configuration

**MiniCore, MightyCore, MegaCore, MajorCore and MicroCore**

Please read official documentation how to configure MC Udude’s Cores:

- Configure “MiniCore”
- Configure “MightyCore”
- Configure “MegaCore”
- Configure “MajorCore”
- Configure “MicroCore”
Debugging

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512B</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ATmega128/A</td>
<td>Atmel AVR</td>
<td>ATMega128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega1280</td>
<td>Atmel AVR</td>
<td>ATMega1280</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1281</td>
<td>Atmel AVR</td>
<td>ATMega1281</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1284</td>
<td>Atmel AVR</td>
<td>ATMega1284</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega16</td>
<td>Atmel AVR</td>
<td>ATMega16</td>
<td>16MHz</td>
<td>16KB</td>
<td>1K</td>
</tr>
<tr>
<td>ATmega164/P/PA</td>
<td>Atmel AVR</td>
<td>ATMega164P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1K</td>
</tr>
<tr>
<td>ATmega168/A</td>
<td>Atmel AVR</td>
<td>ATMega168</td>
<td>16MHz</td>
<td>16KB</td>
<td>1K</td>
</tr>
<tr>
<td>ATmega168/P/PA</td>
<td>Atmel AVR</td>
<td>ATMega168P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1K</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>Atmel AVR</td>
<td>ATMega2560</td>
<td>16MHz</td>
<td>256KB</td>
<td>8K</td>
</tr>
<tr>
<td>ATmega324A</td>
<td>Atmel AVR</td>
<td>ATMega324A</td>
<td>16MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>ATmega324P</td>
<td>Atmel AVR</td>
<td>ATMega324P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>ATmega324PA</td>
<td>Atmel AVR</td>
<td>ATMega324PA</td>
<td>16MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>ATmega328</td>
<td>Atmel AVR</td>
<td>ATMega328</td>
<td>16MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>ATmega328/P/PA</td>
<td>Atmel AVR</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>ATmega48/A</td>
<td>Atmel AVR</td>
<td>ATMega48</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega48/P/PA</td>
<td>Atmel AVR</td>
<td>ATMega48P</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega644/P/PA</td>
<td>Atmel AVR</td>
<td>ATMega644P</td>
<td>16MHz</td>
<td>64KB</td>
<td>4K</td>
</tr>
<tr>
<td>ATmega8/A</td>
<td>Atmel AVR</td>
<td>ATMega8</td>
<td>16MHz</td>
<td>8KB</td>
<td>1K</td>
</tr>
<tr>
<td>ATmega88/A</td>
<td>Atmel AVR</td>
<td>ATMega88</td>
<td>16MHz</td>
<td>8KB</td>
<td>1K</td>
</tr>
<tr>
<td>ATmega88/P/PA</td>
<td>Atmel AVR</td>
<td>ATMega88P</td>
<td>16MHz</td>
<td>8KB</td>
<td>1K</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATtiny13</td>
<td>Atmel AVR</td>
<td>ATTINY13</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>ATtiny13A</td>
<td>Atmel AVR</td>
<td>ATTINY13A</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>Adafruit Bluefruit Micro</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Feather 328P</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Feather 32u4</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>Atmel AVR</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>51B</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/8MHz</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 5V/16MHz</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (USB)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (USB)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Trinket 3V/8MHz</td>
<td>Atmel AVR</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>51B</td>
</tr>
<tr>
<td>Adafruit Trinket 5V/16MHz</td>
<td>Atmel AVR</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>8KB</td>
<td>51B</td>
</tr>
<tr>
<td>Alorium Hin</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Mini Wireless</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduboy</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduboy DevKit</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino BT ATmega168</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega168</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega328</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Esplora</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Industrial 101</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Leonardo ETH</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino LilyPad USB</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>Atmel AVR</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>124KB</td>
<td>8K</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega168</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega8</td>
<td>Atmel AVR</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
<td>1K</td>
</tr>
<tr>
<td>Arduino Nano ATmega168</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1K</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>35.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Yun Mini</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16</td>
</tr>
<tr>
<td>BBC micro:bit V2</td>
<td>Nordic nRF52</td>
<td>NRF52833</td>
<td>64MHz</td>
<td>512KB</td>
<td>64</td>
</tr>
<tr>
<td>BQ ZUM BT-328</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>BitWizard Raspduino</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Calliope mini</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16</td>
</tr>
<tr>
<td>Controllino Maxi</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>24KB</td>
<td>8K</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>24KB</td>
<td>8K</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>24KB</td>
<td>8K</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>32KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64</td>
</tr>
<tr>
<td>Digispark USB</td>
<td>Atmel AVR</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>5.87KB</td>
<td>51</td>
</tr>
<tr>
<td>Engduino 3</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>EnviroDIY Mayfly</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>32</td>
</tr>
<tr>
<td>FYSETC F6 V1.3</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8K</td>
</tr>
<tr>
<td>Generic ATtiny2313</td>
<td>Atmel AVR</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
<td>12</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>Atmel AVR</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
<td>12</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>Atmel AVR</td>
<td>ATTINY25</td>
<td>8MHz</td>
<td>2KB</td>
<td>12</td>
</tr>
<tr>
<td>Generic ATtiny4313</td>
<td>Atmel AVR</td>
<td>ATTINY4313</td>
<td>8MHz</td>
<td>4KB</td>
<td>25</td>
</tr>
<tr>
<td>Generic ATtiny44</td>
<td>Atmel AVR</td>
<td>ATTINY44</td>
<td>8MHz</td>
<td>4KB</td>
<td>25</td>
</tr>
<tr>
<td>Generic ATtiny45</td>
<td>Atmel AVR</td>
<td>ATTINY45</td>
<td>8MHz</td>
<td>4KB</td>
<td>25</td>
</tr>
<tr>
<td>Generic ATtiny46</td>
<td>Atmel AVR</td>
<td>ATTINY46</td>
<td>8MHz</td>
<td>8KB</td>
<td>51</td>
</tr>
<tr>
<td>Generic ATtiny47</td>
<td>Atmel AVR</td>
<td>ATTINY47</td>
<td>8MHz</td>
<td>8KB</td>
<td>51</td>
</tr>
<tr>
<td>Generic ATtiny48</td>
<td>Atmel AVR</td>
<td>ATTINY48</td>
<td>8MHz</td>
<td>8KB</td>
<td>51</td>
</tr>
<tr>
<td>LightBlue Bean</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>LightUp</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Linino One</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>256KB</td>
<td>16</td>
</tr>
<tr>
<td>LinkIt Smart 7688 Duo</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>LoRa32u4II (868-915MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>LowPowerLab MightyHat</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2K</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>LowPowerLab MoteinoMegA</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16</td>
</tr>
<tr>
<td>Microduino Core (Atmega168PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.5KB</td>
<td>1K</td>
</tr>
<tr>
<td>Microduino Core (Atmega168PA@5M,3.3V)</td>
<td>Atmel AVR</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.5KB</td>
<td>1K</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@16M,5V)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Microduino Core USB (Atmega32U4@16M,5V)</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega1284P@16M,5V)</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core+ (Atmega1284P@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>ATMEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic</td>
<td>nRF51</td>
<td>32MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic</td>
<td>nRF51</td>
<td>32MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic</td>
<td>nRF51</td>
<td>32MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic</td>
<td>nRF52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic</td>
<td>nRF52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenEnergyMonitor emonPi</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Original Prusa i3 MK3 Multi Material 2.0 Upgrade</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pololu A-Star 32U4</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prusa RAMBo</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quirkbot</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic</td>
<td>nRF51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic</td>
<td>nRF52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RedBearLab BLE Blend</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RedBearLab BLE Blend 2</td>
<td>Nordic</td>
<td>nRF52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RedBearLab BLE Blend Micro 3.3V/16MHz (overclock)</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RedBearLab BLE Blend Micro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RepRap RAMBo</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SODAQ GaLoRa</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SODAQ Mbili</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SODAQ Moja</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SODAQ Ndogo</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SODAQ Tatu</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 32F417GDISCOVERY</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST 32F417HDISCOVERY</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST</td>
<td>STM32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F765ZG</td>
<td>ST STM32</td>
<td>STM32F765ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8K</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8K</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>STM32L073RZT6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>STM32L152RE</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>STM32L412KBU6</td>
<td>80MHz</td>
<td>132KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
<td>132KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F030DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F030R0T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8K</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VG6T</td>
<td>168MHz</td>
<td>32KB</td>
<td>8K</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F100RBT6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8K</td>
</tr>
<tr>
<td>ST STM8S-DISCOVERY</td>
<td>ST STM8</td>
<td>STM8S105C6T6</td>
<td>12MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (16MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>64KB</td>
<td>4K</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA644</td>
<td>16MHz</td>
<td>64KB</td>
<td>4K</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA644</td>
<td>8MHz</td>
<td>64KB</td>
<td>4K</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA644P</td>
<td>8MHz</td>
<td>64KB</td>
<td>4K</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2K</td>
</tr>
<tr>
<td>SparkFun ATmega128RFAB Ad TI Board</td>
<td>Atmel AVR</td>
<td>ATMEGA128RFAB</td>
<td>16MHz</td>
<td>16KB</td>
<td>2K</td>
</tr>
<tr>
<td>SparkFun Fio 3 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5K</td>
</tr>
<tr>
<td>SparkFun Makey Makey</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.5K</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8K</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3 V/16MHz</td>
<td>Atmel AVR</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5K</td>
</tr>
<tr>
<td>SparkFun Pro Micro 5 V/16/8MHz</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.5K</td>
</tr>
<tr>
<td>SparkFun Qduino Mini</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5K</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2K</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2K</td>
</tr>
<tr>
<td>SpellFinder Spi 2</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>TI Fraunhopper MSP-EXP430FR5739LP</td>
<td>TI MSP430</td>
<td>MSP430F845T</td>
<td>16MHz</td>
<td>15.37KB</td>
<td>1K</td>
</tr>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>TI TIVA</td>
<td>LTM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>TI TIVA</td>
<td>LPM4C12303P</td>
<td>80MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>TI TIVA</td>
<td>LPM4C1294NCPDT</td>
<td>120MHz</td>
<td>1MB</td>
<td>25K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>TI MSP430</td>
<td>MSP430F5529</td>
<td>25MHz</td>
<td>47KB</td>
<td>8K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2311LP</td>
<td>TI MSP430</td>
<td>MSP430FR2311</td>
<td>16MHz</td>
<td>3.75KB</td>
<td>1K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2433LP</td>
<td>TI MSP430</td>
<td>MSP430FR2433</td>
<td>8MHz</td>
<td>15KB</td>
<td>2K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR4133LP</td>
<td>TI MSP430</td>
<td>MSP430FR4133</td>
<td>8MHz</td>
<td>15KB</td>
<td>2K</td>
</tr>
</tbody>
</table>

Continued on next page
### PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI LaunchPad MSP-EXP430FR5969LP</td>
<td>TI MSP430</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>47KB</td>
<td>2K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5994LP</td>
<td>TI MSP430</td>
<td>MSP430FR5994</td>
<td>16MHz</td>
<td>256KB</td>
<td>4K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>TI MSP430</td>
<td>MSP430RF6989</td>
<td>8MHz</td>
<td>47KB</td>
<td>2K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</td>
<td>TI MSP430</td>
<td>MSP430G2231</td>
<td>1MHz</td>
<td>2KB</td>
<td>25</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</td>
<td>TI MSP430</td>
<td>MSP430G2452</td>
<td>16MHz</td>
<td>8KB</td>
<td>25</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2553LP</td>
<td>TI MSP430</td>
<td>MSP430G2553</td>
<td>16MHz</td>
<td>16KB</td>
<td>51</td>
</tr>
<tr>
<td>Talk2 Whisper Node</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2K</td>
</tr>
<tr>
<td>The Things Uno</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.5</td>
</tr>
<tr>
<td>TinyCircuits TinyDuino Processor Board</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>USBasp stick</td>
<td>Atmel AVR</td>
<td>ATMEGA8</td>
<td>12MHz</td>
<td>8KB</td>
<td>1K</td>
</tr>
<tr>
<td>Wicked Device WildFire V2</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>120.00KB</td>
<td>16</td>
</tr>
<tr>
<td>Wicked Device WildFire V3</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16</td>
</tr>
<tr>
<td>XMC1100 Boot Kit</td>
<td>Infineon XMC</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>Infineon XMC</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>Infineon XMC</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>Infineon XMC</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>64KB</td>
<td>16</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>Infineon XMC</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>64KB</td>
<td>16</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>Infineon XMC</td>
<td>XMC1400</td>
<td>48MHz</td>
<td>1.95MB</td>
<td>16</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>Infineon XMC</td>
<td>XMC4200</td>
<td>80MHz</td>
<td>256KB</td>
<td>4K</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>Infineon XMC</td>
<td>XMC4700</td>
<td>144MHz</td>
<td>2.00MB</td>
<td>1.5</td>
</tr>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64</td>
</tr>
<tr>
<td>ftDuino</td>
<td>Atmel AVR</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.5</td>
</tr>
<tr>
<td>nicai-systems BOB3 coding bot</td>
<td>Atmel AVR</td>
<td>ATMEGA88</td>
<td>8MHz</td>
<td>8KB</td>
<td>1K</td>
</tr>
<tr>
<td>nicai-systems NIBO 2 robot</td>
<td>Atmel AVR</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4K</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot</td>
<td>Atmel AVR</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1K</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot</td>
<td>Atmel AVR</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1K</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64</td>
</tr>
<tr>
<td>ublQio Ardhat</td>
<td>Atmel AVR</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2K</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with debugging but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit BLM Badge</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Crickit M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 25 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Expressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 CAN</td>
<td>Atmel SAM</td>
<td>SAME51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>STM32F405RGTE</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M4</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>Atmel SAM</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit MONSTER M4SK</td>
<td>Atmel SAM</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Matrix Portal M4</td>
<td>Atmel SAM</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M0 Expresss</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>Atmel SAM</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M4 AirLift Lite</td>
<td>Atmel SAM</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>Atmel SAM</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>Atmel SAM</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>Atmel SAM</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4 Titanino</td>
<td>Atmel SAM</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Trellis M4</td>
<td>Atmel SAM</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pHkey</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pyBadge AirLift M4</td>
<td>Atmel SAM</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1008KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit pyBadge M4 Express</td>
<td>Atmel SAM</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WSN 1300</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WSN 1310</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Nano 33 BLE</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>960KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F303CC</td>
<td>ST STM32</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BlucDK</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - Samd21</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - Samd21</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel SAM</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32n IoT Uno</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Econode-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Electromath Daisy</td>
<td>ST STM32</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>F407/M1</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FYSE TC S6</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Gnarl-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IotaoP Magnolia</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ItsyBitsy nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH ET LIVE ESP32DevKit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MKR Sharky</td>
<td>ST STM32</td>
<td>STM32WB55CG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192.00KB</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>STM32F103REt6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>Mintronics v2.0</td>
<td>Atmel SAM</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>STM32F405RGt6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>STM32G071RBT6</td>
<td>64MHz</td>
<td>128KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>STM32G431KBt6</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Olimex ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Olimex ESP32-EVB</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Olimex ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OlimexINO-STM32</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>STM32F405RGt6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>PVBStick26 Duino</td>
<td>ST STM32</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PVBStick 26 Pro</td>
<td>ST STM32</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PVBStick Lite 26</td>
<td>ST STM32</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PVBStick Standard 26</td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Particle Xenon</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Picocomix PX-HER0</td>
<td>ST STM32</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>PrntBoard V2</td>
<td>ST STM32</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Raytac MDBT5Q-DX Dongle</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ExpLoRer</td>
<td>Atmel SAM</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>STM32F407ZGt6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RE</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RT4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RT4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103RT6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RT6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103TB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VC6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VD6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VE</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-----</td>
<td>-----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F417CH</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417RG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F417RG</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F417RH</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F423CH</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423RG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F423RG</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F423RH</td>
<td>168MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F446RG</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Seeeduino Femto M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Terminal</td>
<td>Atmel SAM</td>
<td>SAMD51P19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SensorFile.box</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit with Mic</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun 9DOF Razor IMU M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun Qwiic Micro</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD51 Thing Plus</td>
<td>Atmel SAM</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.2MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 25 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>Teensy</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>Teensy</td>
<td>MK64FX512</td>
<td>120MHz</td>
<td>512KB</td>
<td>255.99KB</td>
</tr>
<tr>
<td>Teensy 3.6</td>
<td>Teensy</td>
<td>MK66FX1M0</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Teensy 4.0</td>
<td>Teensy</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>1.94MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy 4.1</td>
<td>Teensy</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>7.75MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>Teensy</td>
<td>MKL26Z64</td>
<td>48MHz</td>
<td>62KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Vint Labs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Waveshare Open1025Z</td>
<td>ST STM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>NFR51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Examples

- Arduino for ASR Microelectronics ASR605x
- Arduino for Atmel AVR
- Arduino for Atmel megaAVR
- Arduino for Atmel SAM
- Arduino for Espressif 32
- Arduino for Espressif 8266
- Arduino for GigaDevice GD32V
- Arduino for Infineon XMC
- Arduino for Intel ARC32
- Arduino for Kendryte K210
- Arduino for Microchip PIC32
- Arduino for Nordic nRF51
- Arduino for Nordic nRF52
• Arduino for ST STM32
• Arduino for ST STM8
• Arduino for Teensy
• Arduino for TI MSP430
• Arduino for TI TIVA
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASR Microelectronics ASR605x</strong></td>
<td>ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.</td>
</tr>
<tr>
<td><strong>Atmel AVR</strong></td>
<td>Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming</td>
</tr>
<tr>
<td><strong>Atmel megaAVR</strong></td>
<td>8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.</td>
</tr>
<tr>
<td><strong>Atmel SAM</strong></td>
<td>Atmel i SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.</td>
</tr>
<tr>
<td><strong>Espressif 32</strong></td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td><strong>Espressif 8266</strong></td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td><strong>GigaDevice GD32V</strong></td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td><strong>Infineon XMC</strong></td>
<td>Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform</td>
</tr>
<tr>
<td><strong>Intel ARC32</strong></td>
<td>ARC embedded processors are a family of 32-bit CPUs that are widely used in SoC devices for storage, home, mobile, automotive, and Internet of Things applications.</td>
</tr>
<tr>
<td><strong>Kendryte K210</strong></td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td><strong>Microchip PIC32</strong></td>
<td>Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!</td>
</tr>
<tr>
<td><strong>Nordic nRF51</strong></td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.</td>
</tr>
<tr>
<td><strong>Nordic nRF52</strong></td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td><strong>ST STM32</strong></td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
<tr>
<td><strong>ST STM8</strong></td>
<td>The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.</td>
</tr>
<tr>
<td><strong>Teensy</strong></td>
<td>Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.</td>
</tr>
<tr>
<td><strong>TI MSP430</strong></td>
<td>MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.</td>
</tr>
</tbody>
</table>
Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

4D Systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4D Systems PICadillo 35T</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>4D Systems gen4 IoT Range</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

AI Thinker

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

AZ-Delivery

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
</tbody>
</table>

Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit BLM Badge</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit Micro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Cricket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Adafruit Feather 328P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Feather 32u4</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 CAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 26 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG16</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512KB</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51P20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit HUZZAH ESP8266</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/8MHz</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/16MHz</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit MONSTER M4SK</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Matrix Portal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>SAMD51J20A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Metro M0 Expresss</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M4 AirLift Lite</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (FTDI)</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (USB)</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (USB)</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4 Titano</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Trellis M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Trinket 3V/8MHz</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512KB</td>
</tr>
<tr>
<td>Adafruit Trinket 5V/16MHz</td>
<td>Atmel AVRX</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>8KB</td>
<td>512KB</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pIRkey</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pyBadge AirLift M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1008KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit pyBadge M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>ItsyBitsy nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

AfroFlight

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>80MHz</td>
<td>32KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Airbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
</tbody>
</table>
Aiyarafun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Alorium Technology

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alorium Hinj</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Amperka

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi Slot</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

Anarduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anarduino MiniWireless</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

April Brother

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>April Brother ESPea32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Arduboy

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduboy</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduboy DevKit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino BT ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------</td>
<td>-------------</td>
<td>---------------</td>
<td>-----------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Dueilanove or Diecimila ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Espora</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Industrial 101</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Leonardo ETH</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino LilyPad USB</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKR WAN 1310</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>124KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino NG or older ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino NG or older ATmega8</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Nano 33 BLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>960KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Nano ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Nano Every</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>47.5KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Uno WiFi Rev2</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>47.5KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Yun Mini</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
</tbody>
</table>
Table 27 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKR Vidor 4000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
</tbody>
</table>

Armed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

Atmel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Generic ATtiny1634</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY1634</td>
<td>8MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Generic ATtiny167</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY167</td>
<td>8MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny2313</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY25</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny261</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY261</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny4313</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY4313</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny43U</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY43U</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny44</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY44</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny441</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY441</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny45</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY45</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny461</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY461</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny48</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY48</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny828</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY828</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny84</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY84</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny841</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY841</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny85</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny861</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY861</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny87</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY87</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny88</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY88</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>USBasp stick</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>12MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
</tbody>
</table>

BBC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>BBC micro:bit V2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52833</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
### BOXTEC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelvePic32</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>48MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HelvePic32</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>48MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HelvePic32 MX270</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX270F256B</td>
<td>48MHz</td>
<td>244KB</td>
<td>62KB</td>
</tr>
<tr>
<td>HelvePic32 Robot</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>244KB</td>
<td>62KB</td>
</tr>
<tr>
<td>HelvePic32 SMD MX270</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>244KB</td>
<td>62KB</td>
</tr>
</tbody>
</table>

### BPI Tech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPI-Bit</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>160MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### BQ

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BQ ZUM BT-328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### BSFrance

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoRa32u4II (868-915MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### BitWizard

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitWizard Raspduino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### BluzDK

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BluzDK</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Calliope

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calliope mini</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>
### ChipKIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB Station</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>240KB</td>
<td>62KB</td>
</tr>
</tbody>
</table>

### Controllino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllino Maxi</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### DFRobot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### DOIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### DSTIKE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Delta

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### DigiStump

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DigiStump Oak</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
# Digilent

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digilent Cerebot 32MX4</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent Cerebot 32MX7</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent OpenScope</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MZ2048EFG124</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Digilent chipKIT Cmod</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX150F128D</td>
<td>40MHz</td>
<td>124KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent chipKIT DP32</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent chipKIT MAX32</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent chipKIT MX3</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX320F128H</td>
<td>80MHz</td>
<td>124KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Digilent chipKIT Pro MX4</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digilent chipKIT Pro MX7</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent chipKIT UNO32</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX320F128H</td>
<td>80MHz</td>
<td>124KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Digilent chipKIT WF32</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX695F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Digilent chipKIT WiFire</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MZ2048ECG100</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Digilent chipKIT uC32</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX340F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>chipKIT WiFire rev. C</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MZ2048EFG100</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

# Digistump

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digispark Pro</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY167</td>
<td>16MHz</td>
<td>14.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digispark Pro (16 MHz)</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY167</td>
<td>16MHz</td>
<td>14.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digispark Pro (64 byte</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY167</td>
<td>16MHz</td>
<td>14.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digispark Pro (32 byte</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY167</td>
<td>16MHz</td>
<td>14.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digispark USB</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>5.87KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>
### Diymore

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Doit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Mx DevKit (ESP8285)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPDuino (ESP-13 Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Dongsen Technology

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Dwengo

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dwenguino</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT90USB646</td>
<td>16MHz</td>
<td>60KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### DycodeX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPectro Core</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### ESP32vn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32vn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### ESPert

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPpresso Lite 1.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPpresso Lite 2.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
### ESPino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Econode

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Econode-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Electronic SweetPeas

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronic SweetPeas ESP320</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Electronut Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### Electrosmith

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Elektor

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elektor Uno R4</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328PB</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Engduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engduino 3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>
EnviroDIY

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnviroDIY Mayfly</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-WROOM-02</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROOM-02</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP8266 ESP-12E</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 1M</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 512k</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07 1MB</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07S</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Generic ESP8285 Module</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 1.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 2.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WifInfo</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

FYSETC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYSETC F6 V1.3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Fred

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## Fubarino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fubarino Mini</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX250F128D</td>
<td>48MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Fubarino SD (1.5)</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Mini 2.0</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>240KB</td>
<td>62KB</td>
</tr>
</tbody>
</table>

## Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Continued on next page
### PlatformIO Documentation, Release 5.0.5a1

**Table 28 – continued from previous page**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F401CD (96k RAM, 384k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM, 128k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 256k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM, 64k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410R8 (32k RAM, 64k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F410R8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM, 128k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM, 256k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM, 256k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 1536k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM, 256k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

**Gimasi**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**HY**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny STM103T</td>
<td>ST ST M32</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>
# Hardkernel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODROID-GO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

# Heltec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec CubeCell Capsule Solar Sensor (HTCC-AC02)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6501</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-1/2AA Node (HTCC-AB02A)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Board (HTCC-AB01)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6501</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Board Plus (HTCC-AB02)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Capsule (HTCC-AC01)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6501</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-GPS (HTCC-AB02S)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Module (HTCC-AM01)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6501</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec CubeCell-Module Plus (HTCC-AM02)</td>
<td>ASR Microelectronics ASR605x</td>
<td>No</td>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Heltec Wifi kit 8</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

# Heltec Automation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec WiFi Kit 32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

# Hornbill

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
ITEAD

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonoff Basic</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff S20</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff SV</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff TH</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

Infineon

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMC1100 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>32KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1400</td>
<td>48MHz</td>
<td>1.95MB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4200</td>
<td>80MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4700</td>
<td>144MHz</td>
<td>2.00MB</td>
<td>1.95MB</td>
</tr>
</tbody>
</table>

Intel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino/Genuino 101</td>
<td>Intel ARC32</td>
<td>No</td>
<td>ARCV2EM</td>
<td>32MHz</td>
<td>152KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

IntoRobot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntoRobot Fig</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Invent One

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invent One</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

IoTaaP

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoTaaP Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### LeafLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB76</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
</tbody>
</table>

### LightUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightUp</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### Linino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linino One</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### LowPowerLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowPowerLab CurrentRanger</td>
<td>Atmel SAM</td>
<td>No</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>LowPowerLab MightyHat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab MoteinoMEGA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### M5Stack

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M5Stack Core ESP32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>M5Stack FIRE</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>6.25MB</td>
</tr>
<tr>
<td>M5Stack GREY ESP32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>M5Stick-C</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### 1.11. Frameworks
### MH-ET Live

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### MXChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Azure IoT Development Kit (MX-Chip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### Magicblocks.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagicBit</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### MakerAsia

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakerAsia Nano32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Makerology

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataStation Mini</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX150F128C</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Malyan

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### MediaTek Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkIt Smart 7688 Duo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

Chapter 1. Contents
## Microchips

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT90CAN128</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT90CAN128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>AT90CAN32</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT90CAN32</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>AT90CAN64</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT90CAN64</td>
<td>16MHz</td>
<td>64KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega128/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega1280</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1281</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1281</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1284</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega1284P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega16</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega1608</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA1608</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega1609</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA1609</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega162</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA162</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA164A</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA164P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168PB</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>256KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega2561</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA2561</td>
<td>16MHz</td>
<td>256KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega32</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega3208</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA3208</td>
<td>16MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega3209</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA3209</td>
<td>16MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega324A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA324A</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA324P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA324PA</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA324PB</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA48</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega4808</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA4808</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ATmega4809</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ATmega48P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA48P</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega48PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA48PB</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega64/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA64</td>
<td>16MHz</td>
<td>64KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega640</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA640</td>
<td>16MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega644/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA644A</td>
<td>16MHz</td>
<td>64KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega644P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>64KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega8/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega808</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA808</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega809</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA809</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega8515</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA8515</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega8535</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA8535</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega88/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA88</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA88P</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA88PB</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny13</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY13</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATtiny13A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY13A</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>ATtiny1604</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY1604</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny1606</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY1606</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny1607</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY1607</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny1614</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY1614</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny1616</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY1616</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny1617</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY1617</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny202</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY202</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny204</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY204</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny212</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY212</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny214</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY214</td>
<td>16MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>ATtiny3216</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY3216</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny3217</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY3217</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATtiny402</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY402</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny404</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY404</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny406</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY406</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny412</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY412</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny414</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY414</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny416</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY416</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny417</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY417</td>
<td>16MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>ATtiny804</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY804</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny806</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY806</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny807</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY807</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny814</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY814</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny816</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY816</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATtiny817</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATTINY817</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>AVR-IoT WG Development Board</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA4808</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>Curiosity Nano ATmega4809</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
<tr>
<td>Xplained Pro ATmega4809</td>
<td>Atmel megaAVR</td>
<td>No</td>
<td>ATMEGA4809</td>
<td>16MHz</td>
<td>48KB</td>
<td>6KB</td>
</tr>
</tbody>
</table>
### Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core (Atmega168PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega168PA@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA168P</td>
<td>8MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core ESP32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>628MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>Microduino Core USB (Atmega32U4@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega1284P@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega1284P@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

### Midatronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKR Sharky</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32WB55CG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192.00KB</td>
</tr>
</tbody>
</table>

### MikroElektronika

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MikroElektronika Clicker 2</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>MikroElektronika Flip N Click MZ</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MZ2048EFH100</td>
<td>252MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

### Netduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>
## NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU 0.9 (ESP-12 Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>NodeMCU 1.0 (ESP-12E Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## NuDuino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NuDuino Quantum</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Nordic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

## OLIMEX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-2-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PRO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PoE</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PoE-ISO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## OROCA

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OROCA EduBot</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### OSHChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB16</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex MOD-WIFI-ESP8266(-DEV)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Olimex PIC32-PINGUINO</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>STM32F440F256H</td>
<td>80MHz</td>
<td>252KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB16</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Onehorse

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onehorse ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### OpenBCI

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenBCI 32bit</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### OpenEnergyMonitor

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenEnergyMonitor emonPi</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### PONTECH

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PONTECH UAV100</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX440F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
## PYBStick

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYBStick26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
<td>128MHz</td>
<td>16KB</td>
<td></td>
</tr>
<tr>
<td>PYBstick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>512KB</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>PYBstick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>96KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PYBstick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>128KB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## PanStamp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanStamp AVR</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

## Particle

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Xenon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

## Piconomix

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
<td>128KB</td>
<td></td>
<td>20KB</td>
</tr>
</tbody>
</table>

## Pinoccio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pinoccio Scout</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA256RFR2</td>
<td>16MHz</td>
<td>248KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Pololu Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pololu A-Star 32U4</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

## Pontech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pontech NoFire</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MZ2048EFG100</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Pontech Quick240</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### PrntrBoard

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrntrBoard V2</td>
<td>STM32</td>
<td>External</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### Prusa 3D

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Prusa i3 MK3 Multi Material 2.0 Upgrade</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Prusa RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### Punch Through

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightBlue Bean</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Pycom Ltd.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pycom GPy</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
</tbody>
</table>

### Qmobot LLP

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qchip</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Quirkbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quirkbot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>
### RAK

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### RUMBA

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Raytac

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raytac MDBT50Q-RX Dongle</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

### RedBearLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/16MHz (overclock)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### RemRam

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>
### RepRap

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepRap RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### ReprapWorld

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minitronics v2.0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### RobotDyn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Pill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### RoboticsBrno

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SG-O

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SODAQ

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ExpLoReR</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ GaLoRa</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Mbili</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Moja</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SODAQ Ndago</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ Tatu</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------</td>
<td>-------</td>
<td>-------------------</td>
<td>-----------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407VE1T6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407VE1T6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Blue STM32F407VE3</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407VE1T6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32G071RBT6</td>
<td>64MHz</td>
<td>128KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32G431KBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32G431KBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32WB55RG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192.00KB</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F413HBT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F030RBT6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F031KBT6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F207ZG1T6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F302RBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F303KBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F303KBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F401RBT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F411RBT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F476ZG</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F476ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F576ZG</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F576ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L031KBT6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L053RBT6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L073RBT6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L152RBT6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L412KBT6</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L432KBT6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L452RBT6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 30 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F0308R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F100RBT6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM8S-DISCOVERY</td>
<td>ST STM8</td>
<td>On-board</td>
<td>STM8S105C6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST STM8S103F3 Breakout Board</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S103F3P6</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ST STM8S105K4T6 Breakout Board</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S105K4T6</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
</tbody>
</table>

#### SainSmart

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

#### Sanguino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanguino ATmega1284p (16MHz)</td>
<td>Atmel</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>Atmel</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>Atmel</td>
<td>On-board</td>
<td>AT-MEGA644</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>Atmel</td>
<td>On-board</td>
<td>AT-MEGA644</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (16 MHz)</td>
<td>Atmel</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>Atmel</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

#### Schirmilabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schirmilabs Eduino WiFi</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
Seeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeeduino Femto M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Terminal</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51P19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SeeedStudio CUI32stem</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Seeeduino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Wio Link</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Wio Node</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

Silicognition

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
# Sipeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------</td>
<td>-------------</td>
<td>--------------</td>
<td>-----------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>Pic32 CUI32-Development Stick</td>
<td>Microchip</td>
<td>No</td>
<td>32MX440F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun 9DoF Razor IMU M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun ATmega128RFA1 Development Board</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA128RFA1</td>
<td>16MHz</td>
<td>16KB</td>
<td>124KB</td>
</tr>
<tr>
<td>SparkFun Blynk Board</td>
<td>Espressif</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun Digital Sandbox</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing</td>
<td>Espressif</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing Dev</td>
<td>Espressif</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun Fio V3 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun Makey Makey</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro Mini 3.3V</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Qduino Mini</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Qwiic Mini</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD51 Thing Plus</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>
### SparkFun Electronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SpellFoundry

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpellFoundry Sleepy Pi 2</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### SweetPea

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SweetPea ESP-210</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
## TI

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI FraunchPad MSP-EXP430FR5739LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>15.37KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>TI TIVA</td>
<td>On-board</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>TI TIVA</td>
<td>On-board</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>TI TIVA</td>
<td>On-board</td>
<td>LPTM4C1294NCP120</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430F5529</td>
<td>25MHz</td>
<td>47KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2311LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR2311</td>
<td>16MHz</td>
<td>3.75KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2433LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR2433</td>
<td>8MHz</td>
<td>15KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR4133LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR4133</td>
<td>8MHz</td>
<td>15KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5969LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5994LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5994</td>
<td>16MHz</td>
<td>256KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2231</td>
<td>1MHz</td>
<td>2KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2452</td>
<td>16MHz</td>
<td>8KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2553LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2553</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
</tbody>
</table>

## TTGO

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T-Watch</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Taida Century

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
### TauLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### Teensy

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teensy 2.0</td>
<td>Teensy</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Teensy 3.0</td>
<td>Teensy</td>
<td>No</td>
<td>MK20DX128</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Teensy 3.1/3.2</td>
<td>Teensy</td>
<td>External</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>Teensy</td>
<td>External</td>
<td>MK64FX512</td>
<td>120MHz</td>
<td>512KB</td>
<td>255.99KB</td>
</tr>
<tr>
<td>Teensy 3.6</td>
<td>Teensy</td>
<td>External</td>
<td>MK66FX1M0</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Teensy 4.0</td>
<td>Teensy</td>
<td>External</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>1.94MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy 4.1</td>
<td>Teensy</td>
<td>External</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>7.75MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>Teensy</td>
<td>External</td>
<td>MKL26Z64</td>
<td>48MHz</td>
<td>62KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Teensy++ 2.0</td>
<td>Teensy</td>
<td>No</td>
<td>AT90USB1286</td>
<td>16MHz</td>
<td>127KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### ThaiEasyElec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ThaiEasyElec ESPino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### The Things Network

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Things Uno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### ThunderPack

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Till Harbaum

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftDuino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>
### TinyCircuits

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyCircuits TinyDuino Processor Board</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### TinyPICO

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyPICO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Tlera Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Turta

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turta IoT Node</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### UBW32

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>UBW32 MX460</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX460F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</td>
</tr>
<tr>
<td>UBW32 MX795</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Unknown

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## VAE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## VCCGND

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## VintLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Wemos

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS D1 R1</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 R2 and mini</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos D1 mini Lite</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos D1 mini Pro</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>16MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Waveshare

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## WeAct

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeAct Black Pill V2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
## Wicked Device

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wicked Device WildFire V2</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>120.00KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## Widora

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Widora AIR</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## WifiDuino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFidualino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Wisen

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk2 Whisper Node</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

## XinaBox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinaBox CW01</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## YeaCreate

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>YeaCreate NSCREEN-32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## chipKIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>chipKIT Lenny</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX270F256D</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
decaWave

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

element14

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element14 chipKIT Pi</td>
<td>Microchip PIC32</td>
<td>No</td>
<td>32MX250F128B</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

makerlab.mx

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altair</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA256RFR2</td>
<td>16MHz</td>
<td>248KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

meteca

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

ng-beacon

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
### nicai-systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>nicai-systems BOB3 coding bot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA88</td>
<td>8MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBO 2 robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### oddWires

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### sduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sduino MB (STM8S208MBT6B)</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S208MBT6</td>
<td>16MHz</td>
<td>128KB</td>
<td>6KB</td>
</tr>
<tr>
<td>sduino UNO (STM8S105K6)</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S105K6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### sino:bit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>u-blox NINA-W10 series</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>2MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
ubIQio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubIQio Ardhat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

1.11.2 CMSIS

**Configuration framework = cmsis**

The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.

For more detailed information please visit vendor site.

## Contents

- Debugging
- Examples
- Platforms
- Boards

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools

### Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in "platformio.ini" *(Project Configuration File)*.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.
On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F723EKT6</td>
<td>216MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>STM32F1765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>STM32F041VGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>STM32F174VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>STM32F142VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>STM32F040VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>STM32L476RT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F411VIT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F476GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F476HIG6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>STM32F030RBT6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>STM32F091RBT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>STM32F302RBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K6</td>
<td>ST STM32</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>STM32F303RBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZI</td>
<td>ST STM32</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>STM32F334RBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>STM32F401RBT6</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>STM32F411RBT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F740ZG</td>
<td>ST STM32</td>
<td>STM32F740ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>STM32F756ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI-Q</td>
<td>ST STM32</td>
<td>STM32H743ZIT6</td>
<td>480MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L402K6</td>
<td>ST STM32</td>
<td>STM32L402K6T6</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>STM32L476RG T6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>STM32L486RG T6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>ST STM32F030DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>STM32L073VZ T6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L152RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F100RBT6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>STM32F7508N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</td>
</tr>
<tr>
<td>STM32H747T-DISCO</td>
<td>ST STM32</td>
<td>STM32H747XH T6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

**External Debug Tools**

Boards listed below are compatible with *Debugging* but **DEPEND ON** external debug probe. They **ARE NOT** READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>STM32F420GRT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>STM32F407VE6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>STM32F303CT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>STM32F401CU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>STM32F401CU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>STM32F407VE6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>F-407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>STM32F407VE6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>STM32F446VE6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>STM32F103RE6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>STM32F405RG6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KI</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>STM32L151RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>STM32G071RB76</td>
<td>64MHz</td>
<td>128KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>STM32G431KBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>STM32G431RB76</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>128KB</td>
<td>128KI</td>
</tr>
<tr>
<td>Olimexino-STM32</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>STM32F405RG6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KI</td>
</tr>
<tr>
<td>PYBstick26 Duino</td>
<td>ST STM32</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBstick 26 Pro</td>
<td>ST STM32</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KI</td>
</tr>
<tr>
<td>PYBstick Lite 26</td>
<td>ST STM32</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBstick Standard 26</td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KI</td>
</tr>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>PrntrBoard V2</td>
<td>ST STM32</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KI</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RHF76052</td>
<td>ST STM32</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32-E407</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32-H407</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>STM32F3073VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RHF76052</td>
<td>ST STM32</td>
<td>STM32F3073VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST STM32F072-VAL</td>
<td>ST STM32</td>
<td>STM32F072BTC6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32F103C4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST STM32F103C6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>ST STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST STM32F103R6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>ST STM32F103RB (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32F103RC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RC6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F103RD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103RF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F103RG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F103T4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST STM32F103T6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>ST STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32F103V8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32F103VB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32F103VC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VC6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F103VD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VD6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103VF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F103VG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F103ZC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZC6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F103ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103ZE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103ZF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZP</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F103ZG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F303CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST STM32F401CB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 32 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F407VG (192k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F407VG T6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192K</td>
</tr>
<tr>
<td>STM32F410CR (32k RAM. 64k Flash)</td>
<td>ST ST32</td>
<td>STM32F410CR</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CRB (32k RAM. 128k Flash)</td>
<td>ST ST32</td>
<td>STM32F410CRB</td>
<td>100MHz</td>
<td>128K</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CR8 (32k RAM. 64k Flash)</td>
<td>ST ST32</td>
<td>STM32F410CR8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM. 128k Flash)</td>
<td>ST ST32</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128K</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512K</td>
<td>256K</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256K</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>1MB</td>
<td>256K</td>
</tr>
<tr>
<td>STM32F412RC (256k RAM. 128k Flash)</td>
<td>ST ST32</td>
<td>STM32F412RC</td>
<td>100MHz</td>
<td>1MB</td>
<td>256K</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM. 128k Flash)</td>
<td>ST ST32</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256K</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320K</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM. 1536k Flash)</td>
<td>ST ST32</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320K</td>
</tr>
<tr>
<td>STM32F413R (320k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F413R</td>
<td>100MHz</td>
<td>1MB</td>
<td>320K</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM. 1536k Flash)</td>
<td>ST ST32</td>
<td>STM32F413R</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320K</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM. 1536k Flash)</td>
<td>ST ST32</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320K</td>
</tr>
<tr>
<td>STM32F423R (320k RAM. 1536k Flash)</td>
<td>ST ST32</td>
<td>STM32F423R</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320K</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM. 256k Flash)</td>
<td>ST ST32</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM. 256k Flash)</td>
<td>ST ST32</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST ST32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192K</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST ST32</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td>640K</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST ST32</td>
<td>STM32F303CT6</td>
<td>72MHz</td>
<td>256K</td>
<td>40KB</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>ST ST32</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192K</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST ST32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST ST32</td>
<td>STM32F103BU6</td>
<td>72MHz</td>
<td>128K</td>
<td>20KB</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>ST ST32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512K</td>
<td>64KB</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST ST32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512K</td>
<td>64KB</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST ST32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512K</td>
<td>64KB</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST ST32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512K</td>
<td>128K</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>ST ST32</td>
<td>STM32F401K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75K</td>
</tr>
</tbody>
</table>

Examples

- CMSIS for ST STM32

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST STM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
</tbody>
</table>
## Boards

**Note:**
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### 1BitSquared

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### 96Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### AfroFlight

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Airbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
### Armed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### Armstrap

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### Avnet Silica

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Diymore

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Electrosmith

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Espotel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### FYSETC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Generic
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F103C4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RC T6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RE T6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103T4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T8 T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103V6 (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103V6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103V8 (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZE T6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401KB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 33 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F410R8 (32k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410R8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM. 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RH (320k RAM. 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM. 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### HY

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### LeafLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
</tbody>
</table>

### MXChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGD</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
## Malyan

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
</tbody>
</table>

## MultiTech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Netduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## PYBStick

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYBSTICK26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBStick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### Piconomix

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### PrntrBoard

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrntrBoard V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### RAK

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### RUMBA

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### RemRam

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

### RobotDyn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### RushUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### ST
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>3DP001VJ Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G071RBT6</td>
<td>64MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RHIF76 052</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST 32F474GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F474NEH6</td>
<td>216MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NIH6</td>
<td>216MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070RB</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302RBT6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152REt6</td>
<td>32MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KBU6</td>
<td>80MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VG16</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073VZ7T6</td>
<td>32MHz</td>
<td>192K</td>
<td></td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RB76</td>
<td>32MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST STM32VLDiscovery</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RB76</td>
<td>24MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>STM32F10C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>STM32F33C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F33VCT6</td>
<td>72MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F7508H6</td>
<td>216MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>STM32H747I-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747XI6</td>
<td>400MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9Z</td>
<td>120MHz</td>
<td>2MB</td>
<td></td>
</tr>
</tbody>
</table>

**SeeedStudio**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512K</td>
<td>192K</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256K</td>
</tr>
</tbody>
</table>
### Semtech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### TauLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### ThunderPack

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### VAE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAkE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### VCCGND

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### Waveshare

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### WeAct

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### 1.11.3 ESP8266 Non-OS SDK

**Configuration** `framework = esp8266-nonos-sdk`

The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions.

For more detailed information please visit vendor site.

#### Contents

- **Examples**
- **Platforms**
- **Boards**

#### Examples

- ESP8266 Non-OS SDK for Espressif 8266

#### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif 8266</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
</tbody>
</table>

#### Boards

**Note:**

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
### 4D Systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4D Systems gen4 IoT Range</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit HUZZAH ESP8266</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Amperka

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi Slot</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Doit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Mx DevKit (ESP8285)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPDuino (ESP-I3 Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### DycodeX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPectro Core</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### ESPert

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPresso Lite 1.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPresso Lite 2.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### ESPino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
## Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-WROOM-02</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif ESP8266 ESP-12E</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 1M</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 512k</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07 1MB</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07S</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Generic ESP8285 Module</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 1.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 2.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WifInfo</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Heltec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec Wifi kit 8</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## ITEAD

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonoff Basic</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff S20</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff SV</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff TH</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Invent One

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invent One</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
### NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU 0.9 (ESP-12 Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>NodeMCU 1.0 (ESP-12E Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olimex MOD-WIFI-ESP8266(-DEV)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Schirmilabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schirmilabs Eduino WiFi</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wio Link</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Wio Node</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun Blynk Board</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing Dev</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### SweetPea

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SweetPea ESP-210</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### ThaiEasyElec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThaiEasyElec ESPino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
### WEMOS

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS D1 R1</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WEMOS D1 R2 and mini</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WEMOS D1 mini Pro</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>16MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### WifiDuino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFiduino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### XinaBox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinaBox CW01</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### 1.11.4 ESP8266 RTOS SDK

**Configuration** *framework* = *esp8266-rtos-sdk*

ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

For more detailed information please visit vendor site.

### Contents

- Examples
- Platforms
- Boards

### Examples

- ESP8266 RTOS SDK for Espressif 8266

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif 8266</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
</tbody>
</table>
## Boards

**Note:**
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### 4D Systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4D Systems gen4 IoD Range</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit HUZZAH ESP8266</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Amperka

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFi Slot</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### Doit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Mx DevKit (ESP8285)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPduino (ESP-13 Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### DycodeX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESpectro Core</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### ESPert

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPpresso Lite 1.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>ESPpresso Lite 2.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
## ESPino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-WROOM-02</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif ESP8266 ESP-12E</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 1M</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 512k</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07 1MB</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-07S</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Generic ESP8285 Module</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 1.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Phoenix 2.0</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WiFiInfo</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Heltec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec Wifi kit 8</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## ITEAD

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonoff Basic</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff S20</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff SV</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Sonoff TH</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
## Invent One

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invent One</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU 0.9 (ESP-12 Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>NodeMCU 1.0 (ESP-12E Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Olimex MOD-WIFI-ESP8266(-DEV)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Schirmilabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schirmilabs Eduino WiFi</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wio Link</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Wio Node</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun Blynk Board</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>SparkFun ESP8266 Thing Dev</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## SweetPea

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SweetPea ESP-210</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
### ThaiEasyElec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThaiEasyElec ESPino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### WEMOS

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS D1 R1</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos D1 R2 and mini</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>WeMos D1 mini Pro</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>16MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### WifiDuino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiFiduino</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### XinaBox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinaBox CW01</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

### 1.11.5 Espressif IoT Development Framework

**Configuration framework = espidf**

ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.

For more detailed information please visit vendor site.

### Contents

- **Tutorials**
- **Configuration**
- **Debugging**
- **Examples**
- **Platforms**
- **Boards**

### Tutorials

- Get started with ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis
Configuration

Note: Starting with ESP-IDF v4.0, a CMake-based build system is used. Different configuration steps are required for ESP-IDF v3.x due to a legacy build system based on GNU Make.

- Configuration for 4.0
- Configuration for 3.0, 3.1, 3.2, 3.3

Each release of Espressif 32 platform uses a specific version of ESP-IDF. The latest version of the platform only supports the latest stable version of the framework.

Warning: ESP-IDF v4.0 projects are not backwards-compatible with ESP-IDF v3.x projects in terms of project configuration process.

Configuration for 4.0

- Project Structure
- ESP-IDF components
- ULP coprocessor programming
- Limitations

The general project configuration (default optimization level, bootloader configuration partition tables, etc) is set in a single file called sdkconfig in the root folder of the project. This configuration file can be modified via a special target called menuconfig (PlatformIO v4.3.0 greater is required):

```
pio run -t menuconfig
```

Warning: ESP-IDF requires some extra tools to be installed in your system in order to build firmware for supported chips. Most of these tools are available in PlatformIO ecosystem as standalone packages, but in order to use configuration tool called menuconfig several additional packages need to be installed on Linux-based systems:

```
libncurses5-dev flex bison
```

More details about required packages can be found in the official ESP-IDF documentation - Standard Setup of Toolchain for Linux.

Project Structure

The ESP-IDF framework requires an unusual project structure because most of the framework configuration is performed by the native for the ESP-IDF build system called CMake.

A typical PlatformIO project for the ESP-IDF framework must have the following structure:
Tip: It’s also possible to use the default ESP-IDF project structure with `main` folder. To specify `main` as the folder with source files use `src_dir` option, for example:

```plaintext
[platformio]
src_dir = main

[env:esp32dev]
platform = espressif32
framework = espidf
board = esp32dev
```

Besides the files related to PlatformIO project, there are several additional ESP-IDF-specific files: the main `CMakeLists.txt`, project-specific `CMakeLists.txt` in `src_dir` and optional default configuration file `sdkconfig.defaults`. `CMakeLists.txt` files enable features supported by the ESP-IDF’s build system, e.g. ULP configuration, adding extra components, etc. A typical `CMakeLists.txt` file in the root folder has the following content:

```plaintext
# The following lines of boilerplate have to be in your project's CMakeLists
# in this exact order for cmake to work correctly
cmake_minimum_required(VERSION 3.16.0)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(project-name)
```

The second `CMakeLists.txt` in `src_dir` is responsible for controlling the build process of the component and its integration into the overall project. The minimal component `CMakeLists.txt` file simply registers the component to the build system using `idf_component_register`:

```plaintext
idf_component_register(SRCS "foo.c" "bar.c")
```

The files specified using `idf_component_register` are used ONLY for generating build configurations, but it’s highly recommended to specify all application source files in order to keep the project compatible with the usual ESP-IDF workflow.

**Warning:** By default PlatformIO expects source files to be located in the `src` folder. At the same time, the default location for source files within the ESP-IDF build system is a special folder with the name `main`. Renaming the main component may require users to manually specify additional dependencies:

```plaintext
idf_component_register(SRCS "main.c" REQUIRES idf::mbedtls)
```

More details in the official ESP-IDF documentation - Renaming main component.

Due to the current limitations of CMake file-based API, there is no way of generating build configuration for source files written in various programming languages if they are not specified in `idf_component_register` command. If your project contains libraries written in languages that differ from the language used for the main application you
need to create an empty file with the desired extension (e.g. *.cpp for C++) in order to force CMake generate build configuration for this language.

**Note:** Build configuration generated for source files specified in `idf_component_register` is also used as the base build environment for project sources (including libraries).

### ESP-IDF components

ESP-IDF modules as modular pieces of standalone code might be useful for structuring reusable code or including third party components that aren’t part of ESP-IDF.

These components contain either a single `CMakeLists.txt` file which controls the build process of the component and its integration into the overall project. An optional `Kconfig` file defines the component configuration options that can be set via `menuconfig`. Some components may also include `Kconfig.projbuild` and `project_include.cmake` files, which are special files for overriding parts of the project. All valid components will be compiled as static libraries and linked to the final firmware. There are two possible ways of adding extra components to PlatformIO project:

- By adding a new component to an optional folder called `components` in the root of your project. This folder will be automatically scanned for valid components.
- Using `EXTRA_COMPONENT_DIRS` option in the root `CMakeLists.txt` file. This option represents a list of extra directories to search for components.

An example of specifying `esp-aws-iot` as an extra component:

```cmake
# The following lines of boilerplate have to be in your project's CMakeLists
# in this exact order for cmake to work correctly
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
list(APPEND EXTRA_COMPONENT_DIRS esp-aws-iot)
project(subscribe_publish)
```

**Warning:** Since `src_dir` is also passed to CMake as an extra component, you should only append to `EXTRA_COMPONENT_DIRS` variable in order not to override the default package.

Since the build may not work correctly if the full path to sources is greater than 250 characters (see `CMAKE_OBJECT_PATH_MAX`) it might be a good idea to keep modules close to the project files.

### ULP coprocessor programming

If you want to take measurements using ADC, internal temperature sensor or external I2C sensors, while the main processors are in deep sleep mode you need to use ULP coprocessor. At the moment ULP can be used only with the *Espressif IoT Development Framework*.

All ULP code, usually written in assembly in files with `.S` extension, must be placed into a separate directory with the name `ulp` in the root folder of your project. So your project structure should look like this:

```
project_dir
    ├── include
```

(continues on next page)
Since PlatformIO uses the code model generated by CMake it’s mandatory to specify ULP source files in CMakeLists.txt as well. An example of typical CMakeLists.txt for ULP:

```cmake
idf_component_register(
    SRCS "ulp_adc_example_main.c"
)
#
# ULP support additions to component CMakeLists.txt.
#
# 1. The ULP app name must be "ulp_main"
set(ulp_app_name ulp_main)
#
# 2. Specify all assembly source files.
# Paths are relative because ULP files are placed into a special directory "ulp"
# in the root of the project
set(ulp_s_sources "../ulp/adc.S")
#
# 3. List all the component source files which include automatically
# generated ULP export file, ${ulp_app_name}.h:
set(ulp_exp_dep_srcs "ulp_adc_example_main.c")
#
# 4. Call function to build ULP binary and embed in project using the argument
# values above.
ulp_embed_binary(${ulp_app_name} ${ulp_s_sources} ${ulp_exp_dep_srcs})
```

See full examples with ULP coprocessor programming:

- https://github.com/platformio/platform-espressif32/tree/develop/examples/espidf-ulp-adc
- https://github.com/platformio/platform-espressif32/tree/develop/examples/espidf-ulp-pulse

More details are located in the official ESP-IDF documentation - ULP coprocessor programming.

### Limitations

At the moment several limitations are present:

- No whitespace characters allowed in project paths. This limitation is imposed by the native ESP-IDF build system. This affects users that have a whitespace in their username or added a whitespace to the project name. As a workaround, it’s recommended to move core_dir to a folder without spaces. For example:

  ```ini
  [platformio]
  core_dir = C:/platformio
  
  [env:esp32dev]
  platform = espressif32
  framework = espidf
  board = esp32dev
  ```

- The src_filter option cannot be used. It’s done to preserve compatibility with existing ESP-IDF projects. List of source files is specified in the project CMakeLists.txt file.
Configuration for 3.0, 3.1, 3.2, 3.3

Support for ESP-IDF v3.x is considered obsolete and is not available in the latest platform releases. Please check the platform release notes to figure out what version of the platform should be installed to use required ESP-IDF version, for example:

```
[env:esp32dev]
; v1.10.0 is the last version that supports ESP-IDF v3.3
platform = espressif32@1.10.0
framework = espidf
board = esp32dev
```

**Project Structure**

Due to limited support of GNU Make build system used in ESP-IDF v3.x, the project configuration depends on a pregenerated file `sdkconfig.h` which contains a list of macro definitions `CONFIG_*`. These definitions describe project settings that will be used for preparing a proper build environment. You can use the default `sdkconfig.h` shipped with the platform or generate a custom one using native ESP-IDF build environment.

A typical PlatformIO project for ESP-IDF v3.x must have the following structure:

```
project_dir
  └── include
  │    └── README
  ├── lib
  │    └── test
  └── src
      └── sdkconfig.h
          └── main.c
      └── platformio.ini
```

**Enable C++ exceptions**

To enable C++ exceptions for *Espressif IoT Development Framework* add `-D PIO_FRAMEWORK_ESP_IDF_ENABLE_EXCEPTIONS` to `build_flags` of “platformio.ini” (Project Configuration File).

See project example with enabled exceptions.

**ULP coprocessor programming**

To use ULP in your project you need to make sure that it’s enabled in the `sdkconfig.h` configuration file. The following two lines must be added:

```
#define CONFIG_ULP_COPROC_ENABLED 1
#define CONFIG_ULP_COPROC_reserve_MEM 1024
```

Usually `CONFIG_ULP_COPROC_reserve_MEM` is already defined in the default `sdkconfig.h` with value 0. You can modify this value to meet your requirements.

All ULP code, usually written in assembly in files with `.S` extension, must be placed into a separate directory with the name `ulp` in the root folder of your project. So your project structure should look like this:
See full examples with ULP coprocessor programming for ESP-IDF v3.x:

- https://github.com/platformio/platform-espressif32/tree/v1.10.0/examples/espidf-ulp-adc
- https://github.com/platformio/platform-espressif32/tree/v1.10.0/examples/espidf-ulp-pulse

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools

#### Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

#### On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

#### External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-dino-32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32n IoT Uno</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Mini</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VinI Labs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

**Examples**

- Espressif IoT Development Framework for Espressif 32
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espres-sif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

AI Thinker

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

AZ-Delivery

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
</tbody>
</table>

Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Aiyarafun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

April Brother

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>April Brother ESPea32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
BPI Tech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPI-Bit</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>160MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

DFRobot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

DOIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

DSTIKE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Dongsen Technology

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

DycodeX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

ESP32vn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32vn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Electronic SweetPeas

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronic SweetPeas ESP320</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Fred

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Hardkernel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODROID-GO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Heltec Automation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec WiFi Kit 32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Hornbill

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## IntoRobot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntoRobot Fig</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
M5Stack

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M5Stack Core ESP32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>M5Stack FIRE</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>6.25MB</td>
</tr>
<tr>
<td>M5Stack GREY ESP32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>M5Stick-C</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

MH-ET Live

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Magicblocks.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagicBit</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

MakerAsia

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakerAsia Nano32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core ESP32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Noduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noduino Quantum</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## PlatformIO Documentation, Release 5.0.5a1

### OLIMEX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PRO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PoE</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-PoE-ISO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### OROCA

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OROCA EduBot</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Onehorse

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onehorse ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Pycom Ltd.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pycom GPy</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
</tbody>
</table>

### Qmobot LLP

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qchip</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SG-O

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Silicognition

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### SparkFun Electronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### TTGO

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T-Watch</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### ThaiEasyElec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### TinyPICO

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyPICO</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Turta

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turta IoT Node</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Unknown

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### VintLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### WEMOS

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Widora

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Widora AIR</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### XinaBox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### YeaCreate

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>YeaCreate NSCREEN-32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### oddWires

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-blox NINA-W10 series</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>2MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
1.11.6 Freedom E SDK

**Configuration**  \textit{framework} = \textit{freedom-e-sdk}

Open Source Software for Developing on the SiFive Freedom E Platform

For more detailed information please visit vendor site.

Contents

- Configuration
- Debugging
- Examples
- Platforms
- Boards

Configuration

Using with FreeRTOS

In order to add FreeRTOS to your project, it must be explicitly specified as an additional framework in the \textit{framework} field of \textit{“platformio.ini” (Project Configuration File)}, for example:

```
[env:freertos]
platform = sifive
framework = freedom-e-sdk, freertos
board = ...
```

**Note:** FreeRTOS RISC-V port for the \textit{SiFive} development platform is distributed as part of \textit{Freedom E SDK} and cannot be used as a standalone framework

Most of the application specific configuration is done in a special file called \textit{FreeRTOSConfig.h} which must be present in each FreeRTOS-based project. Additional settings for build configuration are set in \textit{“platformio.ini” (Project Configuration File)} using the following syntax \texttt{board_build.freertos.*} where \(*\) is an option from the following list:

**FreeRTOS settings**

\begin{itemize}
  \item \texttt{heap_model}
\end{itemize}

Available values: heap_1, heap_2, heap_3, heap_4, heap_5 \texttt{Default: heap_4}

FreeRTOS offers several heap management schemes that range in complexity and features:

- heap_1 – the very simplest, does not permit memory to be freed.
- heap_2 – permits memory to be freed, but does not coalescence adjacent free blocks.
- heap_3 – simply wraps the standard \texttt{malloc()} and \texttt{free()} for thread safety.
- heap_4 – coalescences adjacent free blocks to avoid fragmentation. Includes absolute address placement option.
• heap_5 – as per heap_4, with the ability to span the heap across multiple non-adjacent memory areas.

More information about FreeRTOS Memory Management can be found in the official documentation.

interrupt_handler
Default: FreedomMetal INTERRUPT_HANDLER
The name of a function to be called to handle interrupts

exception_handler
Default: FreedomMetal_EXCEPTION_HANDLER
The name of a function to be called to handle exceptions

mtime_ctrl_addr
Default: 0x2000000
The address of Machine Timer Register mtime

mpu_wrappers
Available values: enable | disable | Default: disable
Use Memory Protection Unit wrappers

Segger SystemView settings

SystemView is a real-time recording and visualization tool for embedded systems that reveals the true runtime behavior of an application.

systemview
Available values: enable | disable | Default: disable
Compile and link Segger SystemView library

sysview_record_enter_isr
Default: SEGGER_SYSVIEW_RecordEnterISR
The name of a function to be called to record interrupt execution.

sysview_record_exit_isr
Default: SEGGER_SYSVIEW_RecordExitISR
The name of a function to be called to record finish of interrupt execution.

sysview_record_exit_isr_to_scheduler
Default: SEGGER_SYSVIEW_RecordExitISRToScheduler
The name of a function to be called to record finish of interrupt when it’s caused by a context switch.

An example of “platformio.ini” (Project Configuration File) with modified heap settings and enabled SystemView feature:

```ini
[env:sifive-hifive1-revb]
platform = sifive
framework = freedom-e-sdk, freertos
board = hifive1-revb
monitor_speed = 115200
; Configure stack size
```
board_build.freedom-e-sdk.heap_size = 0x400
; Configure heap model and enable SystemView
board_build.freertos.heap_model = heap_1
board_build.freertos.systemview = enable

More information about FreeRTOS package for SiFive development platform can be found in the official repository.

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive Rev B</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Examples

- Freedom E SDK for SiFive
PlatformIO Documentation, Release 5.0.5a1

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### SiFive

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Xilinx

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
</tbody>
</table>

1.11.7 FreeRTOS

Configuration `framework = freertos`

FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms.

For more detailed information please visit vendor site.

Contents

- Debugging
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td></td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
</tr>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Examples

- FreeRTOS for CHIPS Alliance
- FreeRTOS for SiFive
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer.
- For more detailed board information please scroll the tables below by horizontally.

Digilent

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>320MHz</td>
<td></td>
<td>16MB</td>
<td>1.16MB</td>
</tr>
</tbody>
</table>

SiFive

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Xilinx

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
</tbody>
</table>
1.11.8 GigaDevice GD32V SDK

**Configuration framework = gd32vf103-sdk**

GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package. For more detailed information please visit vendor site.

Contents

- Debugging
- Examples
- Platforms
- Boards

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

External Debug Tools

Boards listed below are compatible with *Debugging* but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
Examples

- GigaDevice GD32V SDK for GigaDevice GD32V

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigaDevice GD32V</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Sipeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

1.11.9 Kendryte FreeRTOS SDK

Configuration `framework = kendryte-freertos-sdk`

Kendryte SDK with FreeRTOS support

For more detailed information please visit vendor site.
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>

Examples

- Kendryte FreeRTOS SDK for Kendryte K210

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
</tbody>
</table>
Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### Sipeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>

### 1.11.10 Kendryte Standalone SDK

**Configuration** `framework = kendryte-standalone-sdk`

Kendryte Standalone SDK without OS support

For more detailed information please visit vendor site.

### Contents

- Debugging
- Examples
- Platforms
- Boards

### Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - `External Debug Tools`

### Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini”
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sipeed MAIX Bit</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>

Examples

- Kendryte Standalone SDK for Kendryte K210

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by pio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

Sipeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sipeed MAIX Bit</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>
1.11.11 libopencm3

**Configuration framework = libopencm3**

The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-
M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro
EFM32 and others.

For more detailed information please visit vendor site.

### Contents
- Debugging
- Examples
- Platforms
- Boards

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

### Tools & Debug Probes
- **On-Board Debug Tools**
- **External Debug Tools**

### Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by hori-
izontal. You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini”
(Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug
tool below for the further instructions.

### On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external
debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F4212DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F421ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>-----------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>STM32F417VG T6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>STM32F439YIT6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST L476DMW1K</td>
<td>ST STM32</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F469NH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F476IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F476NH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>STM32L475VG T6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>STM32F207GT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>STM32F303RE T6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>STM32F401RE T6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>STM32F411RBT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>STM32F446RE T6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 36 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L051R8</td>
<td>ST STM32</td>
<td>STM32L051R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>STM32L432KC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>STM32L476RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>STM32L486RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>STM32L496ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>STM32L496ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F0308R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM2F3DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>STM2F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>STM32L073VZT6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F100RB1T6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>STM32F7508R8T6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</td>
</tr>
<tr>
<td>STM32F7574-DISCO</td>
<td>ST STM32</td>
<td>STM32F7508R8T6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>STM2F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>STM2F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>TI TIVA</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>TI TIVA</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>TI TIVA</td>
<td>LPTM4C12394NCPTD</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

**External Debug Tools**

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>STM32F405RG6T6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>STM32F407ZG</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------</td>
<td>---------</td>
<td>-----------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>12KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>12KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>STM32G071RBT6</td>
<td>64MHz</td>
<td>128KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>STM32G431KBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>PYBStick26 Duino</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>PYBStick 26 Pro</td>
<td>ST STM32</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>ST STM32</td>
<td>STM32F401ICEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Picomomix PX-HER0</td>
<td>ST STM32</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>PrntBoard V2</td>
<td>ST STM32</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>8KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32J0C-ECV</td>
<td>ST STM32</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F303CB (32k RAM. 128k Flash)</td>
<td>ST ST32</td>
<td>STM32F303CBT6</td>
<td>100MHz</td>
<td>128K</td>
<td>32K</td>
</tr>
<tr>
<td>STM32F303CG (64k RAM. 256k Flash)</td>
<td>ST ST32</td>
<td>STM32F303CGT6</td>
<td>100MHz</td>
<td>256K</td>
<td>64K</td>
</tr>
<tr>
<td>STM32F303CH (96k RAM. 512k Flash)</td>
<td>ST ST32</td>
<td>STM32F303CHT6</td>
<td>100MHz</td>
<td>512K</td>
<td>96K</td>
</tr>
<tr>
<td>STM32F303CI (128k RAM. 1024k Flash)</td>
<td>ST ST32</td>
<td>STM32F303CIT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F303CJ (192k RAM. 2048k Flash)</td>
<td>ST ST32</td>
<td>STM32F303CJT6</td>
<td>100MHz</td>
<td>2048K</td>
<td>256K</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 37 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F411RC (128k RAM. 256k Flash)</td>
<td>ST STIM32</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM. 256k Flash)</td>
<td>ST STIM32</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM. 512k Flash)</td>
<td>ST STIM32</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM. 1024k Flash)</td>
<td>ST STIM32</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM. 512k Flash)</td>
<td>ST STIM32</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM. 1024k Flash)</td>
<td>ST STIM32</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM. 1024k Flash)</td>
<td>ST STIM32</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM. 1536k Flash)</td>
<td>ST STIM32</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM. 1024k Flash)</td>
<td>ST STIM32</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM. 1536k Flash)</td>
<td>ST STIM32</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM. 1024k Flash)</td>
<td>ST STIM32</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM. 512k Flash)</td>
<td>ST STIM32</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM. 1024k Flash)</td>
<td>ST STIM32</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM. 1536k Flash)</td>
<td>ST STIM32</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM. 1536k Flash)</td>
<td>ST STIM32</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM. 256k Flash)</td>
<td>ST STIM32</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM. 512k Flash)</td>
<td>ST STIM32</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STIM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STIM32</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STIM32</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STIM32</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STIM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STIM32</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>VakE v1.0</td>
<td>ST STIM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STIM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STIM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STIM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STIM32</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75K</td>
</tr>
</tbody>
</table>

### Examples

- `libopencm3` for ST STM32
- `libopencm3` for TI TIVA

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST STIM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
<tr>
<td>TI TIVA</td>
<td>Texas Instruments TM4C12x MCUs offer the industry's most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.</td>
</tr>
</tbody>
</table>

### Boards
Note:

- You can list pre-configured boards by *pio boards* command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### 1BitSquared

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### 96Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Argonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### AfroFlight

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Airbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
</tbody>
</table>
## Armed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## Armstrap

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## Avnet Silica

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476JG /STM32L476ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## Diymore

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## Espotel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## FYSETC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Pill F103C8 (128k)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td></td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td></td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RB</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RE</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103T4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103V8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103ZE</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103ZF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103ZG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM, 128 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM, 256 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM, 384 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM, 128 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 256 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 384 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F407VE</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F407VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410R8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F410R8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 38 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F411CE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1.5MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

HY

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

LeafLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
</tbody>
</table>

MXChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Azure IoT Development Kit (MX-Chip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
## Malyan

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
</tbody>
</table>

## MultiTech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Netduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## PYBStick

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYBStick26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBstick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
## Piconomix

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## PntrBoard

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PntrBoard V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## RAK

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## RUMBA

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## RemRam

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

## RobotDyn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

## RushUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## ST

1.11. Frameworks
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>Additional Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>32F723EEDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723EKE6</td>
<td>216MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G071RBT6</td>
<td>64MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431KBT6</td>
<td>170MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VC8T6</td>
<td>84MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NHI6</td>
<td>180MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ2</td>
<td>32MHz</td>
<td>192K</td>
<td></td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302RBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RBT6</td>
<td>100MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F413ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512K</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------</td>
<td>----------------</td>
<td>----------------</td>
<td>-----------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512K</td>
<td>512K</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512K</td>
<td>512K</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZE</td>
<td>216MHz</td>
<td>512K</td>
<td>512K</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192K</td>
<td>192K</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>512K</td>
<td>512K</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
<td>256K</td>
<td>256K</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256K</td>
<td>256K</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256K</td>
<td>256K</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F0308R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256K</td>
<td>256K</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128K</td>
<td>128K</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073VZT6</td>
<td>32MHz</td>
<td>192K</td>
<td>192K</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RBT6</td>
<td>32MHz</td>
<td>128K</td>
<td>128K</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RBT6</td>
<td>24MHz</td>
<td>128K</td>
<td>128K</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256K</td>
<td>256K</td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256K</td>
<td>256K</td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256K</td>
<td>256K</td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
<td>128K</td>
<td>128K</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32H747T-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747XH6</td>
<td>400MHz</td>
<td>2MB</td>
<td>2MB</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td>2MB</td>
</tr>
</tbody>
</table>

**SeeedStudio**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512K</td>
<td>192K</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256K</td>
</tr>
</tbody>
</table>

**Semtech**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256K</td>
<td>32K</td>
</tr>
</tbody>
</table>

1.11. Frameworks
### TI

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>TI</td>
<td>On-board</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>TI</td>
<td>On-board</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>TI</td>
<td>On-board</td>
<td>LPTM4C1294NCPT</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### TauLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### ThunderPack

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### VAE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAkE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### VCCGND

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### Waveshare

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
WeAct

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

rhomb.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

sakura.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

1.11.12 Mbed

Configuration framework = mbed

Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.

For more detailed information please visit vendor site.
Configuration

- Configuration system
  - Mbed OS 6
  - Mbed OS 5 and Mbed 2
- Build profiles
- Ignoring particular components
- Custom Targets

Configuration system

**Warning:** In Mbed OS 6 the list of supported boards was heavily reduced, only officially supported and well maintained targets left. More information in the official explanation. In order to keep legacy projects built on top of Mbed OS 5 compilable, each development platform contains a list of deprecated boards that is used to dynamically select the proper version of Mbed OS (OS5 or OS6).

PlatformIO allows you to customize mbed OS compile time configuration parameters using `mbed_app.json` manifest. It should be placed into the root of your project and located on the same level as “`platformio.ini`” (Project Configuration File).

Configuration is defined using **JSON**. Some examples of configuration parameters:

- The sampling period for a data acquisition application.
- The default stack size for a newly created OS thread.
- The receive buffer size of a serial communication library.
- The flash and RAM memory size of a target board.

See more details in the official ARM Mbed OS Configuration System.

A few PlatformIO-ready projects based on ARM mbed OS which use `mbed_app.json`:

- Freescale Kinetis: mbed-rtos-kvstore
- ST STM32: mbed-rtos-mesh-minimal

**Warning:** On Windows the maximum length for a path (file name and directory route, also known as `MAX_PATH`) — has been defined by 260 characters which can lead to compilation error for some targets. It’s possible to shorten these paths is to install packages in root of any logical disk by specifying `core_dir`.

Mbed OS 6

Using Bare Metal profile

The bare metal profile implements a subset of Mbed OS’s RTOS APIs that are useful in non-threaded applications, such as semaphores (calling the release API from interrupts) and tickers (to set up a recurring interrupt). It does not
include an RTX, and is therefore suitable for applications that do not require complex thread management. Instead of the RTOS’s scheduler, all activities are polled or interrupt-driven.

Useful links:
- Detailed description of bare metal profile
- Porting a target from Mbed OS 2 to Mbed OS 6 bare metal

**Mbed OS 5 and Mbed 2**

PlatformIO allows compiling projects with or without Mbed OS 5. By default, project is built without the OS feature. Most of the framework functionality requires the OS to be enabled. To add the OS feature you can use a special macro definition that needs be added to `build_flags` of “platformio.ini” (Project Configuration File):

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIOFRAMEWORKMBEDRTOSPRESENT</td>
<td>Build the project with enabled rtos</td>
</tr>
</tbody>
</table>

An example of “platformio.ini” (Project Configuration File) with enabled rtos

```
[env:wizwiki_w7500p]
platform = wiznet7500
framework = mbed
board = wizwiki_w7500p
build_flags = -D PIOFRAMEWORKMBEDRTOSPRESENT
```

**Build profiles**

By default, PlatformIO builds your project using `develop` profile which provides optimized firmware size with full error information and allows MCU to go to sleep mode. In the case when default build profile is not suitable for your project there two other profiles `release` and `debug` that can be enabled using special macro definitions. You can change build profile `build_flags` of “platformio.ini” (Project Configuration File):

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBEDBUILDPROFILERELEASE</td>
<td>Release profile (smallest firmware, minimal error info)</td>
</tr>
<tr>
<td>MBEDBUILDPROFILEDEBUG</td>
<td>Debug profile (largest firmware, disabled sleep mode)</td>
</tr>
</tbody>
</table>

More information about differences between build profiles can be found on the official page ARM Mbed OS Build Profiles.

**Ignoring particular components**

In case you don’t need all parts of the framework or you want to reduce the compilation time, you can explicitly exclude folders with redundant sources. For example, to remove cellular, mbedtls and nanostack features from the build process, navigate to `packages_dir` and create a new file `framework-mbed/features/.mbedignore` with the following contents:

```
cellular/*
mbedtls/*
nanostack/*
```

If you want to exclude the entire folder, simply create `.mbedignore` file and add only one symbol * to this file.

1.11. Frameworks
Custom Targets

In case when your board is not officially supported by Mbed you can manually add custom board definitions to your project. First of all, you need to create a special file custom_targets.json in the root folder of your project where you describe your board, for example here is the configuration for NUCLEO-F401RE board:

```json
{
    "NUCLEO_F401RE": {
        "inherits": [
            "MCU_STM32"
        ],
        "supported_form_factors": [
            "ARDUINO",
            "MORPHO"
        ],
        "core": "Cortex-M4F",
        "extra_labels_add": [
            "STM32F4",
            "STM32F401xE"
        ],
        "macros_add": [
            "STM32F401xE"
        ],
        "config": {
            "clock_source": {
                "help": "Mask value : USE_PLL_HSE_EXTC | USE_PLL_HSE_XTAL (need HW patch) | USE_PLL_HSI",
                "value": "USE_PLL_HSE_EXTC|USE_PLL_HSI",
                "macro_name": "CLOCK_SOURCE"
            }
        },
        "detect_code": [
            "0720"
        ],
        "device_has_add": [
            "SERIAL_ASYNCH",
            "FLASH",
            "MPU"
        ],
        "device_name": "STM32F401RE"
    }
}
```

Secondly, you need to add code specific to your target to the src folder of your project. Usually, it’s a good idea to isolate this code in a separate folder and add the path to this folder to build_flags of “platformio.ini” (Project Configuration File):

```
[env:my_custom_board]
platform = nxplpc
framework = mbed
board = my_custom_board
build_flags = -I$PROJECT_SRC_DIR/MY_CUSTOM_BOARD_TARGET
```

Next, you need to inform PlatformIO that there is a new custom board. To do this, you can create boards directory in the root folder of your project and add a board manifest file with your board name, e.g. my_custom_board.json as described here Custom Embedded Boards

After these steps, your project structure should look like this:
More information about adding custom targets can be found on the official page Adding and configuring targets.

See full examples with a custom board:


**Debugging**

*Debugging* – “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools

**Tools & Debug Probes**

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File)*.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

**On-Board Debug Tools**

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100M</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168M</td>
</tr>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>LPC11U24</td>
<td>48M</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>SAMR21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>SAMD21J18A</td>
<td>48M</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>SAML21J18B</td>
<td>48M</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>LPC1768</td>
<td>204M</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>LPC1768</td>
<td>204M</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>NRF51822</td>
<td>64M</td>
</tr>
<tr>
<td>Delta DFCM-NNN40</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32GG990F1024</td>
<td>48M</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32LG990F256</td>
<td>48M</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32WG990F256</td>
<td>48M</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32ZG222F32</td>
<td>24M</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>NXP LPC</td>
<td>LPC4088</td>
<td>120M</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>NXP LPC</td>
<td>LPC4088</td>
<td>120M</td>
</tr>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>MK64F1N1MVLL12</td>
<td>120M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>Freescale Kinetis</td>
<td>MK20D112VLH5</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>MK22FN512VLH12</td>
<td>120M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>MK64F1N1MVLL12</td>
<td>120M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>Freescale Kinetis</td>
<td>MK66F1N2MVMD18</td>
<td>180M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>MK82FN256VLH15</td>
<td>150M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>MKL05Z32VFMB</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>MKL25Z128VLK4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>MKL27Z64VLH4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>Freescale Kinetis</td>
<td>MKL43Z256VLH4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>Freescale Kinetis</td>
<td>MKL46Z256VLH4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>MKW41Z512VHT4</td>
<td>48M</td>
</tr>
<tr>
<td>GAPuino GAP8</td>
<td>RISC-V GAP</td>
<td>GAP8</td>
<td>250M</td>
</tr>
<tr>
<td>JKSot Wallbot BLE</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>LA76DMW1K</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80M</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>LPC11U68</td>
<td>50M</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>LPC824</td>
<td>30M</td>
</tr>
<tr>
<td>Maxim ARM mbed Enabled Development Platform for MAX32600</td>
<td>Maxim 32</td>
<td>MAX32600</td>
<td>24M</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>STM32F439ZIY6</td>
<td>168M</td>
</tr>
<tr>
<td>NXP LPC500-MAX</td>
<td>NXP LPC</td>
<td>LPC812</td>
<td>30M</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>LPC54114J256BD64</td>
<td>100M</td>
</tr>
<tr>
<td>NXP LPCXpresso54008</td>
<td>NXP LPC</td>
<td>LPC54608ET512</td>
<td>180M</td>
</tr>
<tr>
<td>NXP LPCXpresso55569</td>
<td>NXP LPC</td>
<td>LPC55569</td>
<td>150M</td>
</tr>
<tr>
<td>NXP LPCXpresso55569</td>
<td>NXP LPC</td>
<td>MIMXRT1052DVL6B</td>
<td>600M</td>
</tr>
<tr>
<td>NXP LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>LPC11U24</td>
<td>48M</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>LPC1768</td>
<td>96M</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>Nordic nRF5122-mKIT</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit(PCA1000X)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>Nordic nRF520-DK</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84M</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
<td>-------------------</td>
<td>--------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
</tr>
<tr>
<td>SLSTK3401A Pearl Gecko PG1</td>
<td>Silicon Labs EFM32</td>
<td>EFM32PG1B200F256GM48</td>
<td>40MHz</td>
</tr>
<tr>
<td>SLSTK3701A Giant Gecko S1</td>
<td>Silicon Labs EFM32</td>
<td>EFM32GG11B820F2048GL192</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F769NIH6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST B-L475E-1OT01A Discovery kit</td>
<td>ST STM32</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L055R8</td>
<td>ST STM32</td>
<td>STM32L055R8T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>STM32L432KCUC6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>STM32L433RC</td>
<td>80MHz</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80M</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>STM32L486RGT6</td>
<td>80M</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80M</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>STM32L496ZGT6P</td>
<td>80M</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48M</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F051R8T6</td>
<td>48M</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F303VCT6</td>
<td>72M</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168M</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L476JG</td>
<td>80M</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>STM32L476JG</td>
<td>80M</td>
</tr>
<tr>
<td>STM32H747I-DISCO</td>
<td>ST STM32</td>
<td>STM32H747XIH6</td>
<td>400M</td>
</tr>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168M</td>
</tr>
<tr>
<td>Seeed Arch Pro</td>
<td>Nordic LPC</td>
<td>LPC1768</td>
<td>96M</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>STM32F439V1</td>
<td>180M</td>
</tr>
<tr>
<td>Switch Science mbed HRM1017</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>Nordic nRF51</td>
<td>LPC1114FN28</td>
<td>48M</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>Nordic nRF51</td>
<td>LPC824</td>
<td>16M</td>
</tr>
<tr>
<td>Switch Science mbed TY51822r3</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>VNG VBLUN051</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>WIZwiki-W7500</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>LPC1768</td>
<td>96M</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>STM32F437VG</td>
<td>180M</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>STM32F439ZIY6</td>
<td>168M</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with Debugging but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>NXP LPC</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>DipCortex M3</td>
<td>NXP LPC</td>
<td>LPC1347</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>NXP LPC</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100M</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL82Z</td>
<td>Freescale Kinetis</td>
<td>MKL8Z128VLK7</td>
<td>96MHz</td>
<td>128KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>Freescale Kinetis</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetis</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 41 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX32620FTHR</td>
<td>Maxim 32</td>
<td>MAX32620FTHR</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Maxim Health Sensor Platform</td>
<td>Maxim 32</td>
<td>MAX32620</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Maxim Wireless Sensor Node Demonstrator</td>
<td>Maxim 32</td>
<td>MAX32610</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>NA Mote72</td>
<td>ST STM32</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>NXP LPC</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11C24</td>
<td>NXP LPC</td>
<td>LPC11C24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U34</td>
<td>NXP LPC</td>
<td>LPC11U34</td>
<td>48MHz</td>
<td>40KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U37</td>
<td>NXP LPC</td>
<td>LPC11U37</td>
<td>48MHz</td>
<td>128KB</td>
<td>10KB</td>
</tr>
<tr>
<td>NXP LPCXpresso1549</td>
<td>NXP LPC</td>
<td>LPC1549</td>
<td>72MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Olimexino-STM32</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>NXP LPC</td>
<td>LPC11U24</td>
<td>50MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>Teensy</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>y5 LPC11U35 mbug</td>
<td>NXP LPC</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

Examples

- Mbed for Atmel SAM
- Mbed for Freescale Kinetis
- Mbed for Maxim 32
- Mbed for Nordic nRF51
- Mbed for Nordic nRF52
- Mbed for NXP i.MX RT
- Mbed for NXP LPC
- Mbed for RISC-V GAP
- Mbed for Silicon Labs EFM32
- Mbed for ST STM32
- Mbed for Teensy
- Mbed for WIZNet W7500
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel SAM</td>
<td>Atmel</td>
</tr>
<tr>
<td>Freescale Kinetis</td>
<td>Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.</td>
</tr>
<tr>
<td>Maxim 32</td>
<td>Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.</td>
</tr>
<tr>
<td>Nordic nRF51</td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.</td>
</tr>
<tr>
<td>Nordic nRF52</td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td>NXP i.MX RT</td>
<td>The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.</td>
</tr>
<tr>
<td>NXP LPC</td>
<td>The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.</td>
</tr>
<tr>
<td>RISC-V GAP</td>
<td>GreenWaves GAP8 IoT application processor enables the cost-effective development, deployment and autonomous operation of intelligent sensing devices that capture, analyze, classify and act on the fusion of rich data sources such as images, sounds or vibrations.</td>
</tr>
<tr>
<td>Silicon Labs EFM32</td>
<td>Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.</td>
</tr>
<tr>
<td>ST STM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
<tr>
<td>Teensy</td>
<td>Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.</td>
</tr>
<tr>
<td>WIZ-Net W7500</td>
<td>The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things</td>
</tr>
</tbody>
</table>

## Boards

Note:
- You can list pre-configured boards by *pio boards* command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
### 96Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### AppNearMe

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroNFCBoard</td>
<td>NXP LPC</td>
<td>No</td>
<td>LPC11U34</td>
<td>48MHz</td>
<td>48KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

### Atmel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Avnet Silica

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### BBC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### CQ Publishing

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>
Delta

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Delta DFCM-NNN40</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Elektor Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

Embedded Artists

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

Espotel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
## Freescale

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK20DX128VLH5</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK22FN512VLH12</td>
<td>120MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK66FN2M0VMD</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL05Z32VF</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128V</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL27Z64V</td>
<td>48MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL43Z256V</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL46Z256V</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL82Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL82Z128V</td>
<td>96MHz</td>
<td>128KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKW41Z512V</td>
<td>48MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## GHI Electronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBuino</td>
<td>NXP LPC</td>
<td>No</td>
<td>LPC11U24</td>
<td>50MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

## Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Pill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>
GreenWaves Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAPuino GAP8</td>
<td>RISC-V GAP</td>
<td>On-board</td>
<td>GAP8</td>
<td>250MHz</td>
<td>64MB</td>
<td>8MB</td>
</tr>
</tbody>
</table>

JKSoft

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>JKSof Wallbot BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>nRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Maxim

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX32620FTHR</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32620FTHR</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>MAX32625MBED</td>
<td>Maxim 32</td>
<td>No</td>
<td>MAX32625</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
<tr>
<td>MAX32625NEXPAQ</td>
<td>Maxim 32</td>
<td>No</td>
<td>MAX32625</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
<tr>
<td>MAX32625PICO</td>
<td>Maxim 32</td>
<td>No</td>
<td>MAX32625</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
<tr>
<td>Maxim ARM mbed Enabled Development Platform for MAX32600</td>
<td>Maxim 32</td>
<td>On-board</td>
<td>MAX32600</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Maxim Health Sensor Platform</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32620</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Maxim MAX32630FTHR Application Platform</td>
<td>Maxim 32</td>
<td>No</td>
<td>MAX32630</td>
<td>96MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Maxim Wireless Sensor Node Demonstrator</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32610</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Micromint

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
<td>204MHz</td>
<td>8MB</td>
<td>264KB</td>
</tr>
</tbody>
</table>

MikroElektronika

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
### MultiTech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTS Dragonfly</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### NGX Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### NXP

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
<td>50MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>LPCXpresso11C24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11C24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>LPCXpresso11U34</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U34</td>
<td>48MHz</td>
<td>40KB</td>
<td>8KB</td>
</tr>
<tr>
<td>LPCXpresso11U37</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U37</td>
<td>48MHz</td>
<td>128KB</td>
<td>10KB</td>
</tr>
<tr>
<td>LPC800-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>LPCXpresso1549</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1549</td>
<td>72MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>LPCXpresso54114</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54114J256BD64</td>
<td>100MHz</td>
<td>256KB</td>
<td>192KB</td>
</tr>
<tr>
<td>LPCXpresso54608</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54608ET512</td>
<td>180MHz</td>
<td>512KB</td>
<td>200KB</td>
</tr>
<tr>
<td>LPCXpresso55S69</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC55S69</td>
<td>150MHz</td>
<td>640KB</td>
<td>320KB</td>
</tr>
<tr>
<td>i.MX RT1050 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1052DVL6B4</td>
<td>600MHz</td>
<td>64MB</td>
<td>512KB</td>
</tr>
<tr>
<td>LPC11U24</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>mbed LPC11U24</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
Nordic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Outrageous Circuits

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outrageous Circuits mBaino</td>
<td>NXP LPC</td>
<td>No</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

RedBearLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

RushUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

ST
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>ST32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F769NIH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F030R8</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F031K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F042K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F070RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F103RB</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F303K8</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F334R8</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F446ZET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F474ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F474ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L031K6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L053R8</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L152RET6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L432KCU6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L476RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L486RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks

665
### Table 42 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZG76P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4R5ZI76</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>ST STM32F030DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F100RBT6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STM32H747T-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747XH6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

**SeeedStudio**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Arch GPRS V2</td>
<td>NXP LPC</td>
<td>No</td>
<td>LPC11U37</td>
<td>48MHz</td>
<td>128KB</td>
<td>10KB</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Arch Pro</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439V1</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Seeed Xadow M0</td>
<td>NXP LPC</td>
<td>No</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

**Semtech**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**Sigma Delta Technologies**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT32620B</td>
<td>Maxim 32</td>
<td>No</td>
<td>MAX32620IWG</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>SDT32625B</td>
<td>Maxim 32</td>
<td>No</td>
<td>MAX32625ITK</td>
<td>96MHz</td>
<td>512KB</td>
<td>160KB</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
### Silicon Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32GG990F1024</td>
<td>48MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32LG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32WG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32ZG222F32</td>
<td>24MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SLSTK3401A Pearl Gecko PG1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32PG1B200F256</td>
<td>40MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SLSTK3701A Giant Gecko SI</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
<td>240MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
<td>240MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### Smeshlink

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smeshlink xbed LPC1768</td>
<td>NXP LPC</td>
<td>No</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Solder Splash Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DipCortex M3</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1347</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>50MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### Switch Science

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch Science mbed HRM1017</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1114FN28</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Switch Science mbed TY51822r3</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
Teensy

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>Teensy</td>
<td>External</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

VNG

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNG VBLUNO51</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

WIZNet

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZwiki-W7500</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500ECO</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500P</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
</tbody>
</table>

rhomb.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

sakura.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### y5 design

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>y5 LPC11U35 mbug</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### 1.11.13 Nuclei SDK

Configuration framework = nuclei-sdk

Open Source Software Development Kit for the Nuclei N/NX processors

For more detailed information please visit vendor site.

Contents

- Debugging
- Examples
- Platforms
- Boards

**Debugging**

Debugging - “1-click” solution for debugging with a zero configuration.
• Tools & Debug Probes
  – On-Board Debug Tools
  – External Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>Nuclei</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HummingBird Evaluation Kit</td>
<td>Nuclei</td>
<td>HUMMINGBIRD</td>
<td>5MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>Nuclei</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>Nuclei</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Examples

• Nuclei SDK for Nuclei

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuclei</td>
<td>Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.</td>
</tr>
</tbody>
</table>
Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

GigaDevice

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Nuclei

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>Nuclei</td>
<td>On-board</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HummingBird Evaluation Kit</td>
<td>Nuclei</td>
<td>On-board</td>
<td>HUMMINGBIRD</td>
<td>5MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

1.11.14 PULP OS

Configuration `framework = pulp-os`

PULP is a silicon-proven Parallel Ultra Low Power platform targeting high energy efficiencies. The platform is organized in clusters of RISC-V cores that share a tightly-coupled data memory.

For more detailed information please visit vendor site.

### Contents

- Debugging
- Examples
- Platforms
- Boards

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAPino GAP8</td>
<td>RISC-V GAP</td>
<td>GAP8</td>
<td>250MHz</td>
<td>64MB</td>
<td>8MB</td>
</tr>
</tbody>
</table>

Examples

- PULP OS for RISC-V GAP

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC-V GAP</td>
<td>GreenWaves GAP8 IoT application processor enables the cost-effective development, deployment and autonomous operation of intelligent sensing devices that capture, analyze, classify and act on the fusion of rich data sources such as images, sounds or vibrations.</td>
</tr>
</tbody>
</table>

Boards

**Note:**

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

GreenWaves Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAPino GAP8</td>
<td>RISC-V GAP</td>
<td>On-board</td>
<td>GAP8</td>
<td>250MHz</td>
<td>64MB</td>
<td>8MB</td>
</tr>
</tbody>
</table>
1.11.15 Pumbaa

Configuration `framework = pumbaa`

Pumbaa is Python on top of Simba. The implementation is a port of MicroPython, designed for embedded devices with limited amount of RAM and code memory.

For more detailed information please visit vendor site.

Contents

- Examples
- Platforms
- Boards

Examples

- Pumbaa for Espressif 32

Platforms

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakerAsia Nano32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

1.11.16 Shakti SDK

Configuration `framework = shakti-sdk`

A software development kit for developing applications on Shakti class of processors

For more detailed information please visit vendor site.
Contents

- Debugging
- Examples
- Platforms
- Boards

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Arty A7-100: Artix-7 FPGA Development Board</td>
<td>Shakti</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
<tr>
<td>Parashu on Artix-7 100T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Pinaka on Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Vajra on Arty A7-100: Artix-7 FPGA Development Board</td>
<td>Shakti</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
</tbody>
</table>
Examples

- Shakti SDK for Shakti

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakti</td>
<td>Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

Xilinx

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Arty A7-100: Artyx-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
<tr>
<td>Parashu on Artix-7 100T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Pinaka on Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Vajra on Arty A7-100: Arty-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
</tbody>
</table>

1.11.17 Simba

Configuration `framework = simba`

Simba is an RTOS and build framework with aims to make embedded programming easy and portable

For more detailed information please visit vendor site.

Contents

- Debugging
- Examples
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools
  - External Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

### On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>Atmel AVR</td>
<td>AT-MEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>Atmel AVR</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Atmel AVR</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Seeeduino</td>
<td>Atmel AVR</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with Debugging but **DEPEND ON** external debug probe. They **ARE NOT READY** for debugging. Please click on board name for the further details.
### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel AVR</td>
<td>Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.</td>
</tr>
<tr>
<td>Atmel SAM</td>
<td>Atmel</td>
</tr>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Espressif 8266</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
</tbody>
</table>

### Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

### Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit HUZZAH ESP8266</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
## Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

## Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-WROOM-02</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif ESP8266 ESP-12E</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Espressif Generic ESP8266 ESP-01 512k</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## Invent One

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invent One</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>

## MakerAsia

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MakerAsia Nano32</td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU 0.9 (ESP-12 Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>NodeMCU 1.0 (ESP-12E Module)</td>
<td>Espressif 8266</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
</tbody>
</table>
1.11.18 SIWI GSM Software Development Kit for MT6261 and MT2503 SoC

Configuration framework = siwisdk
Arduino Wiring-based Framework For SiWi and Quectel GSM Modules
For more detailed information please visit vendor site.

1.11.19 Standard Peripheral Library

Configuration framework = spl
The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.
For more detailed information please visit vendor site.

Contents

- Examples
- Debugging
- Examples
- Platforms
- Boards

Examples

All project examples are located in PlatformIO repository Examples for SPL framework.
- Blink

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools
  - External Debug Tools
Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>84MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM8S-DISCOVERY</td>
<td>ST STM8</td>
<td>STM8S105C6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

Examples

- Standard Peripheral Library for ST STM32
- Standard Peripheral Library for ST STM8
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST STM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
<tr>
<td>ST STM8</td>
<td>The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

1BitSquared

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Armstrap

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F427VT16</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F401RB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>
**RushUp**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

**ST**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM8S-DISCOVERY</td>
<td>ST STM8</td>
<td>On-board</td>
<td>STM8S105C6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST STM8S103F3 Breakout Board</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S103F3P6</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ST STM8S105K4T6 Breakout Board</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S105K4T6</td>
<td>16MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

**sduino**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sduino MB (STM8S208MBT6B)</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S208MBT6</td>
<td>16MHz</td>
<td>128KB</td>
<td>6KB</td>
</tr>
<tr>
<td>sduino UNO (STM8S105K6)</td>
<td>ST STM8</td>
<td>No</td>
<td>STM8S105K6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### 1.11.20 STM32Cube

**Configuration** `framework = stm32cube`

STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency

For more detailed information please visit [vendor site](#).
Platforms

Boards

Tutorials

- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

Using with STM32CubeMX

At the moment there is no seamless integration with projects generated by STM32CubeMX tool. Instead, a small cross-platform Python application called stm32pio can be used to create and update PlatformIO projects from STM32CubeMX .ioc files. It uses STM32CubeMX to generate a HAL-framework-based code and alongside creates PlatformIO project with compatible parameters to stick them both together.

More details about stm32pio tool can be found in the official repository.

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

- On-Board Debug Tools
- External Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

On-Board Debug Tools

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>32F723E DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------</td>
<td>--------------</td>
<td>-----------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td></td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>STM32F439ZIY6</td>
<td></td>
<td>168MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F476GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F476NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F769NIH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>STM32F030RBT6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>STM32F302RGT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>STM32F401RBT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>STM32F411RBT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>STM32F446REI6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>STM32F746ZET6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 43 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H745ZI-Q</td>
<td>ST STM32</td>
<td>STM32H745ZIT6</td>
<td>480MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L055R8</td>
<td>ST STM32</td>
<td>STM32L055R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>STM32L073RZT6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>STM32L412KBU6</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48Mhz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F051R8T6</td>
<td>48Mhz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168Mhz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>STM32L073VZT6</td>
<td>32Mhz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L152RBT6</td>
<td>32Mhz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F100RBT6</td>
<td>24Mhz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>STM32L476JG</td>
<td>80Mhz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>STM32F750N8H6</td>
<td>216Mhz</td>
<td>64KB</td>
<td>340KB</td>
</tr>
<tr>
<td>STM32H747I-DISCO</td>
<td>ST STM32</td>
<td>STM32H747XIH6</td>
<td>400Mhz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168Mhz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>STM32F439VI</td>
<td>180Mhz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100Mhz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>STM32F437VG</td>
<td>180Mhz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>STM32F439ZIY6</td>
<td>168Mhz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

External Debug Tools

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>STM32F415RGT</td>
<td>168Mhz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168Mhz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168Mhz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>STM32F412CG</td>
<td>100Mhz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>STM32F427VIT6</td>
<td>168Mhz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>STM32F411CE</td>
<td>100Mhz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168Mhz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>72Mhz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168Mhz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Econode-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>M200V2</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>STM32F405RG76</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>STM32L151RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>STM32G071RB76</td>
<td>64MHz</td>
<td>128KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>STM32G431KB76</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>STM32G431RB76</td>
<td>170MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Olimexino-STM32</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>STM32F405RG76</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>PYBSTICK26 Duino</td>
<td>ST STM32</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBStick 26 Pro</td>
<td>ST STM32</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>ST STM32</td>
<td>STM32F401UCEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>ST STM32</td>
<td>STM32F411UCEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------</td>
<td>-----------</td>
<td>------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>PtnrBoard V2</td>
<td>ST STM32</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>STM32G0316</td>
<td>64MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32J0C-EVAL</td>
<td>ST STM32</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>ST STM32</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F0T2-EVAL</td>
<td>ST STM32</td>
<td>STM32F0T2VBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>ST STM32</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RGT6</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103T4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103V8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (64k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VFT6</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VGT6</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZFT6</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F103ZGT6</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 44 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F407VE</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F407VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F407VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F407VG</td>
<td>168MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410R8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F410R8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>STM32F103TB6</td>
<td>72MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>ST STM32</td>
<td>STM32F446ET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75K</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

Examples

- STM32Cube for ST STM32
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ST STM32</strong></td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
</tbody>
</table>

Boards

**Note:**
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

**1BitSquared**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

**96Boards**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

**Adafruit**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

**AfroFlight**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>
### Airbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
</tbody>
</table>

### Armed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### Armstrap

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### Avnet Silica

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Diymore

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Econode

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Econode-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Electrosmith

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### Espotel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### FYSETC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103TF (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103V8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VC6T6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 45 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F1032F (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F1032F</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F1032G (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F1032G</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CB</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F405RG (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VE</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM. 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM. 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM. 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM. 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

HY
## LeafLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
</tbody>
</table>

## MXChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Azure IoT Development Kit (MX-Chip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

## Malyan

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
</tbody>
</table>

## MultiTech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Netduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
## Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## PYBStick

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYBSTICK26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBstick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PYBstick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBstick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## Piconomix

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## PrntrBoard

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrntrBoard V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

## RAK

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## RUMBA

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### RemRam

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

### RobotDyn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### RushUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### ST

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723EK6</td>
<td>216MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VT6</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G071RBTE6</td>
<td>64MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431KBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G474RE6</td>
<td>170MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F411EDISSCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F4291DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F4691DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NITH6</td>
<td>180MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F7691DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NITH6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td></td>
</tr>
</tbody>
</table>

Continued on
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VG</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AG</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VG</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R</td>
<td>48MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K</td>
<td>48MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K</td>
<td>48MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070R</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072R</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091R</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103R</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207Z</td>
<td>120MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302R</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303R</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303Z</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401R</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410R</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411R</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412Z</td>
<td>100MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413Z</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429Z</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439Z</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446R</td>
<td>180MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446E</td>
<td>180MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722Z</td>
<td>216MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746Z</td>
<td>216MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F756Z</td>
<td>216MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767Z</td>
<td>216MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H743Z</td>
<td>400MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo H745ZI-Q</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H745Z</td>
<td>480MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K</td>
<td>32MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K</td>
<td>32MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053R</td>
<td>32MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073R</td>
<td>32MHz</td>
<td>192KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152R</td>
<td>32MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412K</td>
<td>80MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L432K</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433R</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452R</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RG</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RG</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496Z</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496Z</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4R5Z</td>
<td>120MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F0308R</td>
<td>48MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F051R</td>
<td>48MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>----------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VG16</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073VZT6</td>
<td>32MHz</td>
<td>192KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RB16</td>
<td>32MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F100RB16</td>
<td>24MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F7308-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F7471-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747XIH6</td>
<td>400MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
</tr>
</tbody>
</table>

**SeeedStudio**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

**Semtech**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**TauLabs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

**ThunderPack**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
## Tiera Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

## VAE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## VCCGND

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Waveshare

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## WeAct

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## rhomb.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## sakura.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST</td>
<td>On-board</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

1.11.21 WD-Firmware

**Configuration** framework = wd-riscv-sdk

The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features.

For more detailed information please visit vendor site.

Contents

- Debugging
  - Examples
  - Platforms
  - Boards

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- Tools & Debug Probes
  - On-Board Debug Tools

Tools & Debug Probes

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
### On-Board Debug Tools

Boards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RVfpga: Digilent Nexys A7</strong></td>
<td>CHIPS Alliance</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
</tbody>
</table>

#### Examples

- WD-Firmware for CHIPS Alliance

#### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHIPS Alliance</strong></td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
</tbody>
</table>

#### Boards

**Note:**
- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer.
- For more detailed board information please scroll the tables below by horizontally.

#### Digilent

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RVfpga: Digilent Nexys A7</strong></td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
</tbody>
</table>

### 1.11.22 WiringPi

**Configuration** `framework = wiringpi`

WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry Pi. It’s designed to be familiar to people who have used the Arduino ‘wiring’ system.

For more detailed information please visit [vendor site](#).
Examples

- WiringPi for Linux ARM

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux ARM</td>
<td>Linux ARM is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS X, Linux ARM) you can build native application for Linux ARM platform.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi 1 Model B</td>
<td>Linux ARM</td>
<td>No</td>
<td>BCM2835</td>
<td>700MHz</td>
<td>512MB</td>
<td>512MB</td>
</tr>
<tr>
<td>Raspberry Pi 2 Model B</td>
<td>Linux ARM</td>
<td>No</td>
<td>BCM2836</td>
<td>900MHz</td>
<td>1GB</td>
<td>1GB</td>
</tr>
<tr>
<td>Raspberry Pi 3 Model B</td>
<td>Linux ARM</td>
<td>No</td>
<td>BCM2837</td>
<td>1200MHz</td>
<td>1GB</td>
<td>1GB</td>
</tr>
<tr>
<td>Raspberry Pi Zero</td>
<td>Linux ARM</td>
<td>No</td>
<td>BCM2835</td>
<td>1000MHz</td>
<td>512MB</td>
<td>512MB</td>
</tr>
</tbody>
</table>

1.11.23 Zephyr RTOS

Configuration `framework = zephyr`

The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.

For more detailed information please visit vendor site.
Tutorials

- Zephyr and Nordic nRF52-DK: debugging, unit testing, project analysis
- Enabling PlatformIO and Zephyr on custom hardware

Configuration

- Project Structure
- Devicetree overlays
- Embedding files at compile time
- Zephyr modules
- Limitations

Project Structure

**Warning:** Zephyr framework currently requires Python 3.4 or later.

Zephyr framework requires an unusual project structure because most of the framework configuration is performed by the native for Zephyr build system called CMake.

**Note:** Since PlatformIO is able to generate CMake-based projects for certain IDEs, Zephyr-related files are moved to a separate folder in order to avoid conflicts between project files. That requires users to specify relative paths to source files in CMakeLists.txt.

A typical PlatformIO project for Zephyr framework must have the following structure:

```
project_dir
  |--- include
  |--- src
  |     |--- main.c
  |--- zephyr
  |     |--- prj.conf
  |     |--- CMakeLists.txt
  |--- platformio.ini
```
Besides files related to PlatformIO project, there is an additional folder \texttt{zephyr} that contains Zephyr-specific files \texttt{CMakeLists.txt} and \texttt{prj.conf}:

\texttt{CMakeLists.txt} file enables features supported by Zephyr’s build system, e.g. board-specific kernel configuration files. A typical \texttt{CMakeLists.txt} file has the following content:

\begin{verbatim}
# Boilerplate code, which pulls in the Zephyr build system.
cmake_minimum_required(VERSION 3.13.1)
include($ENV{ZEPHYR_BASE}/cmake/app/boilerplate.cmake NO_POLICY_SCOPE)
project(my_zephyr_app)
# Add your source file to the "app" target. This must come after the boilerplate
# code, which defines the target. Note relative path to source file
target_sources(app PRIVATE ../src/main.c)
\end{verbatim}

The files specified in \texttt{target_sources} are used \textbf{ONLY} for generating build configurations, but it’s highly recommended to specify all application source files in order to keep the project compatible with the usual Zephyr workflow.

Due to the current limitations of CMake file-based API, there is no way to generate build configuration for source files written in various programming languages if they are not specified in \texttt{target_sources} command. If your project contains libraries written in languages that differ from the language used for the main application you need to create an empty file with desired extension (e.g. \texttt{*.cpp} for C++) in order to force CMake generate build configuration for this language.

\textbf{Note:} Build configuration generated for source files specified in \texttt{target_sources} is also used as the base build environment for project sources (including libraries).

\texttt{prj.conf} file sets application-specific values for one or more kernel configuration options. These application settings are merged with board-specific settings to produce a kernel configuration.

\section*{Devicetree overlays}

Zephyr applications can use overlay files to enable a peripheral that is disabled by default, select a sensor on the board for an application specific purpose, etc. This makes it possible to reconfigure the kernel and device drivers without modifying source code. There are several ways to set \texttt{.overlay} files:

\begin{itemize}
  \item Using \texttt{DTC_OVERLAY_FILE} variable in the \texttt{CMakeLists.txt} file, before including Zephyr’s \texttt{boilerplate.cmake} file. (Recommended)
  \item Using a \texttt{boards/<BOARD>.overlay} file in the \texttt{zephyr} folder, for the current board
  \item Using a \texttt{<BOARD>.overlay} file in the \texttt{zephyr} folder.
\end{itemize}

\textbf{Warning:} PlatformIO board names may differ from Zephyr targets, which means that to help the build system automatically pick up \texttt{.overlay} file, the \texttt{<BOARD>} name in \texttt{<BOARD>.overlay} file must have the same name as specified in the official Zephyr board list.

\section*{Embedding files at compile time}

In case your \texttt{CMakeLists.txt} relies on using \texttt{generate_inc_*} functions that are used for generating and compressing individual files (for example certificates for secure connections) you need to configure your PlatformIO project accordingly using the following structure:
Where `apps-cert.der` and `apps-key.der` are the files you want to embed to your project at the compile time.

**Zephyr modules**

**Note:** PlatformIO automatically installs several default modules used with Zephyr framework including modules that implement silicon vendor Hardware Abstraction Layers (HALs).

Zephyr modules are externally maintained packages that allow using well-established and mature code created by third party developers.

These modules contain either a single `module.yml` file or `CMakeLists.txt` and `Kconfig` files that describe how to build and configure them. You can specify paths to additional directories with source code, `Kconfig`, etc. using `ZEPHYR_EXTRA_MODULES` at the top of your project's `CMakeLists.txt` file, for example:

```cmake
# Additional modules
set(ZEPHYR_EXTRA_MODULES "path/to-zephyr-custom-module" [...])

# Boilerplate code, which pulls in the Zephyr build system.
cmake_minimum_required(VERSION 3.13.1)
include($ENV{ZEPHYR_BASE}/cmake/app/boilerplate.cmake NO_POLICY_SCOPE)
project(my_zephyr_app)

# Add your source file to the "app" target. This must come after
# the boilerplate code, which defines the target.
target_sources(app PRIVATE ../src/main.c)
```

Since the build may not work correctly if the full path to sources is greater than 250 characters (see `CMAKE_OBJECT_PATH_MAX`) it might be a good idea to keep modules close to the project configuration files (e.g. in `zephyr` folder) in form of a git submodule.

**Warning:** Make sure the `ZEPHYR_EXTRA_MODULES` variable is set before including the boilerplate file, as shown above.

**Limitations**

At the moment several limitations are present:

- The minimum supported version of Python is 3.4
- No whitespace characters allowed in project paths.
- OpenThread module is not supported
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

- **Tools & Debug Probes**
  - On-Board Debug Tools
  - External Debug Tools

**Tools & Debug Probes**

Supported debugging tools are listed in “Debug” column. For more detailed information, please scroll table by horizontal. You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions.

**On-Board Debug Tools**

Boards listed below have on-board debug probe and ARE READY for debugging! You do not need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>32F723E DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Arm SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Arm SAM</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Arm SAM</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>BL652 Development Kit</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32WG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ElectronutLabs Blip</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Electronut Labs Papyrus</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>MK22FN512VHL12</td>
<td>120MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>MK64F1N0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>MKL25Z128VLK4</td>
<td>48MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>MKW41Z512VHL4</td>
<td>48MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>LPC11U68</td>
<td>50MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>LPC54114J256BD64</td>
<td>100MHz</td>
<td>256KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

Continued on next page.
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NXP LPCXpresso55S16</td>
<td>NXP LPC</td>
<td>LPC55S16</td>
<td>150MHz</td>
<td>256KB</td>
<td>96KB</td>
</tr>
<tr>
<td>NXP i.MX RT1010 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>MIMXRT1011DAE5A</td>
<td>500MHz</td>
<td>64KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1015 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>MIMXRT1015DAF5A</td>
<td>500MHz</td>
<td>96KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1020 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>MIMXRT1021DA5A</td>
<td>500MHz</td>
<td>64MB</td>
<td>256KB</td>
</tr>
<tr>
<td>NXP i.MX RT1050 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>MIMXRT1052DVL6B</td>
<td>600MHz</td>
<td>64MB</td>
<td>512KB</td>
</tr>
<tr>
<td>NXP i.MX RT1060 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>MIMXRT1062DVL6A</td>
<td>600MHz</td>
<td>64MB</td>
<td>1MB</td>
</tr>
<tr>
<td>NXP i.MX RT1064 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>MIMXRT1064DVL6A</td>
<td>600MHz</td>
<td>4MB</td>
<td>1MB</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit(PCA1000X)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429HDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F429ZT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469HDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F474GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F474N6H6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F479HDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F479NIH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L496AG16</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>STM32F072RB6T6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo H745ZI-Q</td>
<td>ST STM32</td>
<td>STM32H745ZIT6</td>
<td>480MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>STM32L053RBT6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>STM32L476RG6A</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 47 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32H747F-DISCO</td>
<td>ST STM32</td>
<td>STM32H7471IH6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>VNG VBLUNO51</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>reel_board</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

**External Debug Tools**

Boards listed below are compatible with Debugging but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>96Boards Nitrogen</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitty M4</td>
<td>Atmel SAM</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>Freescale Kinetis</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetis</td>
<td>MK64F1N0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>Holyiot YJ-16019</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>STM32G071RBT6</td>
<td>64MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Olimexino-STM32</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>Particle Argon</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
</tr>
<tr>
<td>Particle Boron</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 48 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Particle Xenon</strong></td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
</tr>
<tr>
<td><strong>Ruuvi Tag</strong></td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td><strong>SEGGER IP Switch Board</strong></td>
<td>Freescale Kinetis</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
<tr>
<td><strong>STEVAL-FCU001V1 Flight controller unit evaluation board</strong></td>
<td>ST STM32</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>STM32-E407</strong></td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td><strong>STM32-H407</strong></td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td><strong>STM3210C-EVAL</strong></td>
<td>ST STM32</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>STM32373C-EVAL</strong></td>
<td>ST STM32</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>STM32F072-EVAL</strong></td>
<td>ST STM32</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>Seeeduino XIAO</strong></td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>Waveshare BLE400</strong></td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>WeAct BlackPill V2.0 (BlackPill F411CE)</strong></td>
<td>ST STM32</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
</tbody>
</table>

#### Examples

- Zephyr RTOS for Atmel SAM
- Zephyr RTOS for CHIPS Alliance
- Zephyr RTOS for Freescale Kinetis
- Zephyr RTOS for Nordic nRF51
- Zephyr RTOS for Nordic nRF52
- Zephyr RTOS for NXP i.MX RT
- Zephyr RTOS for NXP LPC
- Zephyr RTOS for SiFive
- Zephyr RTOS for Silicon Labs EFM32
- Zephyr RTOS for ST STM32
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel SAM</td>
<td>Atmel</td>
</tr>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
<tr>
<td>Freescale Kinetis</td>
<td>Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.</td>
</tr>
<tr>
<td>Nordic nRF51</td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.</td>
</tr>
<tr>
<td>Nordic nRF52</td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td>NXP i.MX RT</td>
<td>The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.</td>
</tr>
<tr>
<td>NXP LPC</td>
<td>The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
<tr>
<td>Silicon Labs EFM32</td>
<td>Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.</td>
</tr>
<tr>
<td>ST STM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
</tbody>
</table>

## Boards

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll the tables below by horizontally.
## 96Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Argonkey (STEVAL-MKI1187V1)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427V1E6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Nitrogen</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Adafruit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F-405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>ATT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
## Atmel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## BBC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## Digilent

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td></td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
</tr>
</tbody>
</table>

## ElectronutLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElectronutLabs Blip</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ElectronutLabs Papyr</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

## Freescale

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK22FN512VHL12</td>
<td>120MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>1280MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
<td>526KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128VLT4</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKW41Z512VHT4</td>
<td>48MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

---

1.11. Frameworks
### Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Holyiot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holyiot YJ-16019</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### Laird Connectivity

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL652 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### Makerdiary

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### MikroElektronika

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

---

712 Chapter 1. Contents
### NXP

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
<td>50MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54114J256BD64</td>
<td>100MHz</td>
<td>256KB</td>
<td>192KB</td>
</tr>
<tr>
<td>NXP LPCXpresso55S16</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC55S16</td>
<td>150MHz</td>
<td>256KB</td>
<td>96KB</td>
</tr>
<tr>
<td>NXP i.MX RT1010 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1011DAE5A</td>
<td>500MHz</td>
<td>64KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1015 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1015DAF5A</td>
<td>500MHz</td>
<td>96KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1020 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1021DAG5A</td>
<td>500MHz</td>
<td>64MB</td>
<td>256KB</td>
</tr>
<tr>
<td>NXP i.MX RT1050 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1052DVL6B</td>
<td>600MHz</td>
<td>64MB</td>
<td>512KB</td>
</tr>
<tr>
<td>NXP i.MX RT1060 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1062DVL6A</td>
<td>600MHz</td>
<td>64MB</td>
<td>1MB</td>
</tr>
<tr>
<td>NXP i.MX RT1064 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1064DVL6A</td>
<td>600MHz</td>
<td>64MB</td>
<td>1MB</td>
</tr>
</tbody>
</table>

### Nordic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit(PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
## PHYTEC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>reel_board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

## Particle

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Argon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Boron</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Xenon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

## RedBearLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Ruuvi

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruvi Tag</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## SEGGER

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGGER IP Switch Board</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

## ST

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723IET6</td>
<td>216MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G071RBT6</td>
<td>64MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page...
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ST 32F411E DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST 32F429I DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST 32F469I DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST 32F746G DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST 32F769I DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F769NIH6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST 32L476G DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST 32L496G DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST B-L475E-IOT01A Discovery kit</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST DISCO-L072CZ-LRWAN1</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Discovery F072RB</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F030R8</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F070RB</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F091RC</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F103RB</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F207ZG</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F302RR</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F303RE</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F334R8</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F340RE</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F340RGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F411RE</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F412ZG</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F413ZH</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F429ZI</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F446RE</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F446RGT6</td>
<td>180MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F756ZG</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F756ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo F767ZI</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo H743ZI-Q</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo L053R8</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo L073RZ</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo L432KC</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L432KCUC6</td>
<td>80MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo L476RG</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L476RGTC6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo L496ZG</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST Nucleo L4R5ZI</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td><strong>ST STM32F0 DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST STM32F3 DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td><strong>ST STM32F4 DISCOVERY</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td><strong>STEV-FCU001V1 Flight controller unit evaluation board</strong></td>
<td>ST STIM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td><strong>STM3210C-EVAL</strong></td>
<td>ST STIM32</td>
<td>External</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td><strong>STM32F373C-EVAL</strong></td>
<td>ST STIM32</td>
<td>External</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td><strong>STM32F072-EVAL</strong></td>
<td>ST STIM32</td>
<td>External</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td><strong>STM32H747I-DISCO</strong></td>
<td>ST STIM32</td>
<td>On-board</td>
<td>STM32H747XIH6</td>
<td>400MHz</td>
<td>2MB</td>
<td></td>
</tr>
</tbody>
</table>

**Seeed**

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeeduino XIAO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
## SiFive

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## Silicon Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs</td>
<td>On-board</td>
<td>EFM32WG990F28</td>
<td>84MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs</td>
<td>On-board</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

## SparkFun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## VNG

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNG VBLUNO51</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Waveshare

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## WeAct

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
1.12 Boards

Rapid Embedded Development, Continuous and IDE integration in a few steps with PlatformIO thanks to built-in project generator for the most popular embedded boards and IDE.

Note:

- You can list pre-configured boards by `pio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

### 1.12.1 Aceinna IMU

#### Aceinna Low Cost RTK

**Contents**

- Aceinna Low Cost RTK
  - Hardware
  - Configuration
  - Uploading
  - Debugging

**Hardware**

Platform **Aceinna IMU**: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F469NIH6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>384KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Aceinna</td>
</tr>
</tbody>
</table>

**Configuration**

Please use LowCostRTK ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:LowCostRTK]
platform = aceinna_imu
board = LowCostRTK
```

You can override default Aceinna Low Cost RTK settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `LowCostRTK.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
**Uploading**

Aceinna Low Cost RTK supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```ini
[env:LowCostRTK]
platform = aceinna_imu
board = LowCostRTK
upload_protocol = stlink
```

**Debugging**

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Aceinna Low Cost RTK has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Aceinna OpenIMU 300**
Hardware

Platform Aceinna IMU: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F405RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Aceinna</td>
</tr>
</tbody>
</table>

Configuration

Please use OpenIMU300 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:OpenIMU300]
platform = aceinna_imu
board = OpenIMU300
```

You can override default Aceinna OpenIMU 300 settings per build environment using board_*** option, where *** is a JSON object path from board manifest OpenIMU300.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:OpenIMU300]
platform = aceinna_imu
board = OpenIMU300

; change microcontroller
board_build.mcu = stm32f405rg

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Aceinna OpenIMU 300 supports the next uploading protocols:

- blackmagic
- jlink
- stlink
Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:OpenIMU300]
platform = aceinna imu
board = OpenIMU300
upload_protocol = stlink
```

**Debugging**

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Aceinna OpenIMU 300 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Aceinna OpenIMU 300ZA**

**Contents**

- *Aceinna OpenIMU 300ZA*
  - Hardware
  - Configuration
  - Uploading
  - Debugging

**Hardware**

Platform *Aceinna IMU*: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.
### Microcontroller Specifications

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F405RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Aceinna</td>
</tr>
</tbody>
</table>

### Configuration

Please use OpenIMU300ZA ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:OpenIMU300ZA]
platform = aceinna_imu
board = OpenIMU300ZA
```

You can override default Aceinna OpenIMU 300ZA settings per build environment using `board_***` option, where *** is a JSON object path from board manifest OpenIMU300ZA.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:OpenIMU300ZA]
platform = aceinna_imu
board = OpenIMU300ZA

; change microcontroller
board_build.mcu = stm32f405rg

; change MCU frequency
board_build.f_cpu = 120000000L
```

### Uploading

Aceinna OpenIMU 300ZA supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:OpenIMU300ZA]
platform = aceinna_imu
board = OpenIMU300ZA

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Aceinna OpenIMU 300ZA does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Aceinna OpenIMU 330**

**Contents**

- *Aceinna OpenIMU 330*
  - Hardware
  - Configuration
  - Uploading
  - Debugging

**Hardware**

Platform *Aceinna IMU*: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L431CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Aceinna</td>
</tr>
</tbody>
</table>

**Configuration**

Please use OpenIMU330 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:OpenIMU330]
platform = aceinna_imu
board = OpenIMU330
```

You can override default Aceinna OpenIMU 330 settings per build environment using board_*** option, where *** is a JSON object path from board manifest OpenIMU330.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

Aceinna OpenIMU 330 supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:OpenIMU330]
platform = aceinna_imu
board = OpenIMU330
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Aceinna OpenIMU 330 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td><strong>Yes</strong></td>
</tr>
</tbody>
</table>

Aceinna OpenIMU 330ZA
Hardware

Platform **Aceinna IMU**: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F469IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>384KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Aceinna</td>
</tr>
</tbody>
</table>

Configuration

Please use OpenRTK ID for **board** option in “platformio.ini” (Project Configuration File):

```ini
[env:OpenRTK]
platform = aceinna_imu
board = OpenRTK
```

You can override default Aceinna OpenIMU 330ZA settings per build environment using **board_*** option, where *** is a JSON object path from board manifest `OpenRTK.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:OpenRTK]
platform = aceinna_imu
board = OpenRTK

; change microcontroller
board_build.mcu = stm32f469IG

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

Aceinna OpenIMU 330ZA supports the next uploading protocols:

- blackmagic
- jlink
- stlink
Default protocol is **stlink**

You can change upload protocol using `upload_protocol` option:

```ini
[env:OpenRTK]
platform = aceinna_imu
board = OpenRTK
upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Aceinna OpenIMU 330ZA does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Aceinna OpenRTK330L

#### Contents

- *Aceinna OpenRTK330L*
  - Hardware
  - Configuration
  - Uploading
  - Debugging

#### Hardware

Platform *Aceinna IMU*: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.
### Configuration

Please use OpenRTK330L ID for `board` option in “platformio.ini” (*Project Configuration File)*:

```ini
[env:OpenRTK330L]
platform = aceinna_imu
board = OpenRTK330L
```

You can override default Aceinna OpenRTK330L settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `OpenRTK330L.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:OpenRTK330L]
platform = aceinna_imu
board = OpenRTK330L

; change microcontroller
board_build.mcu = stm32f469IG

; change MCU frequency
board_build.f_cpu = 180000000L
```

### Uploading

Aceinna OpenRTK330L supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:OpenRTK330L]
platform = aceinna_imu
board = OpenRTK330L

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Aceinna OpenRTK330L does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

1.12.2 ASR Microelectronics ASR605x

Heltec CubeCell Capsule Solar Sensor (HTCC-AC02)

Contents

- Heltec CubeCell Capsule Solar Sensor (HTCC-AC02)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform ASR Microelectronics ASR605x: ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
<td>Heltec</td>
</tr>
</tbody>
</table>

Configuration

Please use cubecell_capsule_solar_sensor ID for board option in “platformio.ini” (Project Configuration File):

```
[env:cubecell_capsule_solar_sensor]
platform = asrmicro650x
board = cubecell_capsule_solar_sensor
```
You can override default Heltec CubeCell Capsule Solar Sensor (HTCC-AC02) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `cubecell_capsule_solar_sensor.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:cubecell_capsule_solar_sensor]
platform = asrmicro650x
board = cubecell_capsule_solar_sensor

; change microcontroller
board_build.mcu = asr6502

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Debugging

*Debugging* currently does not support Heltec CubeCell Capsule Solar Sensor (HTCC-AC02) board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Heltec CubeCell-1/2AA Node (HTCC-AB02A)

#### Contents

- Heltec CubeCell-1/2AA Node (HTCC-AB02A)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *ASR Microelectronics ASR605x*: ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR6502</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
<td>Heltec</td>
</tr>
</tbody>
</table>
Configuration

Please use `cubecell_node` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:cubecell_node]
platform = asrmicro650x
board = cubecell_node
```

You can override default Heltec CubeCell-1/2AA Node (HTCC-AB02A) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cubecell_node.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cubecell_node]
platform = asrmicro650x
board = cubecell_node

; change microcontroller
board_build.mcu = asr6502

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

`Debugging` currently does not support Heltec CubeCell-1/2AA Node (HTCC-AB02A) board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Heltec CubeCell-Board (HTCC-AB01)

Contents

- Heltec CubeCell-Board (HTCC-AB01)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **ASR Microelectronics ASR605x**: ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ASR6501</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec</td>
</tr>
</tbody>
</table>

Configuration

Please use `cubecell_board` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:cubecell_board]
platform = asrmicro650x
board = cubecell_board
```

You can override default Heltec CubeCell-Board (HTCC-AB01) settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `cubecell_board.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:cubecell_board]
platform = asrmicro650x
board = cubecell_board

; change microcontroller
board_build.mcu = asr6501

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

*Debugging* currently does not support Heltec CubeCell-Board (HTCC-AB01) board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Heltec CubeCell-Board Plus (HTCC-AB02)
Contents

- Heltec CubeCell-Board Plus (HTCC-AB02)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform ASR Microelectronics ASR605x: ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoc 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ASR6502</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec</td>
</tr>
</tbody>
</table>

Configuration

Please use `cubecell_board_plus` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:cubecell_board_plus]
platform = asrmicro650x
board = cubecell_board_plus
```

You can override default Heltec CubeCell-Board Plus (HTCC-AB02) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `cubecell_board_plus.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:cubecell_board_plus]
platform = asrmicro650x
board = cubecell_board_plus

; change microcontroller
board_build.mcu = asr6502

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

Debugging currently does not support Heltec CubeCell-Board Plus (HTCC-AB02) board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Heltec CubeCell-Capsule (HTCC-AC01)

Contents

- Heltec CubeCell-Capsule (HTCC-AC01)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform ASR Microelectronics ASR605x: ASR Microelectronics ASR605x series is highly intergrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ASR6501</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec</td>
</tr>
</tbody>
</table>

Configuration

Please use cubecell_capsule ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:cubecell_capsule]
platform = asrmicro650x
board = cubecell_capsule
```

You can override default Heltec CubeCell-Capsule (HTCC-AC01) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest cubecell_capsule.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cubecell_capsule]
platform = asrmicro650x
board = cubecell_capsule

; change microcontroller
board_build.mcu = asr6501
```

(continues on next page)
; change MCU frequency
board_build.f_cpu = 48000000L

Debugging

Debugging currently does not support Heltec CubeCell-Capsule (HTCC-AC01) board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Heltec CubeCell-GPS (HTCC-AB02S)

Contents

- Heltec CubeCell-GPS (HTCC-AB02S)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform ASR Microelectronics ASR605x: ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ASR602</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec</td>
</tr>
</tbody>
</table>

Configuration

Please use `cubecell_gps` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default Heltec CubeCell-GPS (HTCC-AB02S) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cubecell_gps.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
platform = asrmicro650x
board = cubecell_gps

; change microcontroller
board_build.mcu = asr6502

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Debugging**

*Debugging* currently does not support Heltec CubeCell-GPS (HTCC-AB02S) board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Heltec CubeCell-Module (HTCC-AM01)**

**Contents**

- *Heltec CubeCell-Module (HTCC-AM01)*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *ASR Microelectronics ASR605x*: ASR Microelectronics ASR605x series is highly intergrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.
PlatformIO Documentation, Release 5.0.5a1

Microcontroller | ASR6501  
Frequency       | 48MHz   
Flash           | 128KB   
RAM             | 16KB    
Vendor          | Heltec  

### Configuration

Please use `cubecell_module` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:cubecell_module]
platform = asrmicro650x
board = cubecell_module
```

You can override default Heltec CubeCell-Module (HTCC-AM01) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cubecell_module.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cubecell_module]
platform = asrmicro650x
board = cubecell_module

; change microcontroller
board_build.mcu = asr6501

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Debugging

*Debugging* currently does not support Heltec CubeCell-Module (HTCC-AM01) board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

#### Heltec CubeCell-Module Plus (HTCC-AM02)

**Contents**

- *Heltec CubeCell-Module Plus (HTCC-AM02)*
  - Hardware
  - Configuration

1.12. Boards 735
Hardware

Platform *ASR Microelectronics ASR605x*: ASR Microelectronics ASR605x series is highly integrated and ultra low power SoC based on the PSoC 4000 series MCU (ARM Cortex M0+ Core) and Semtech SX1262 transceiver.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ASR6502</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec</td>
</tr>
</tbody>
</table>

Configuration

Please use `cubecell_module_plus` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:cubecell_module_plus]
platform = asrmicro650x
board = cubecell_module_plus
```

You can override default Heltec CubeCell-Module Plus (HTCC-AM02) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `cubecell_module_plus.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cubecell_module_plus]
platform = asrmicro650x
board = cubecell_module_plus

; change microcontroller
board_build.mcu = asr6502

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

*Debugging* currently does not support Heltec CubeCell-Module Plus (HTCC-AM02) board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
1.12.3 Atmel AVR

AT90CAN128

Contents

- AT90CAN128
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT90CAN128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use AT90CAN128 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:AT90CAN128]
platform = atmelavr
board = AT90CAN128
```

You can override default AT90CAN128 settings per build environment using board_*** option, where *** is a JSON object path from board manifest AT90CAN128.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:AT90CAN128]
platform = atmelavr
board = AT90CAN128

; change microcontroller
board_build.mcu = at90can128

; change MCU frequency
board_build.f_cpu = 16000000L
```
Debugging

Debugging currently does not support AT90CAN128 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

AT90CAN32

Contents

- AT90CAN32
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT90CAN32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use AT90CAN32 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:AT90CAN32]
platform = atmelavr
board = AT90CAN32
```

You can override default AT90CAN32 settings per build environment using board_*** option, where *** is a JSON object path from board manifest AT90CAN32.json. For example, board_build.mcu, board_build.f_cpu, etc.
Debugging

Debugging currently does not support AT90CAN32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

AT90CAN64

Contents

- AT90CAN64
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT90CAN64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>
Configuration

Please use AT90CAN64 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:AT90CAN64]
platform = atmelavr
board = AT90CAN64
```

You can override default AT90CAN64 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest AT90CAN64.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:AT90CAN64]
platform = atmelavr
board = AT90CAN64

; change microcontroller
board_build.mcu = at90can64

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support AT90CAN64 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega128/A

Contents

- ATmega128/A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency, and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

**Configuration**

Please use ATmega128 ID for **board** option in “platformio.ini” *(Project Configuration File):*

```
[env:ATmega128]
platform = atmelavr
board = ATmega128
```

You can override default ATmega128/A settings per build environment using **board_*** option, where *** is a JSON object path from board manifest ATmega128.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega128]
platform = atmelavr
board = ATmega128

; change microcontroller
board_build.mcu = atmega128

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” *(Project Configuration File)*.

ATmega128/A has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega1280

Contents

- ATmega1280
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1280</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega1280 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:ATmega1280]
platform = atmelavr
board = ATmega1280
```

You can override default ATmega1280 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega1280.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega1280]
platform = atmelavr
board = ATmega1280

; change microcontroller
```

(continues on next page)
PlatformIO Documentation, Release 5.0.5a1

```
board_build.mcu = atmega1280

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

ATmega1280 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## ATmega1281

### Contents

- ATmega1281
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *Atmel AVR:* Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1281</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega1281 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATmega1281]
platform = atmelavr
board = ATmega1281
```

You can override default ATmega1281 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `ATmega1281.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega1281]
platform = atmelavr
board = ATmega1281

; change microcontroller
board_build.mcu = atmega1281

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega1281 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega1284

Contents

- ATmega1284
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega1284 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega1284]
platform = atmelavr
board = ATmega1284
```

You can override default ATmega1284 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega1284.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega1284]
platform = atmelavr
board = ATmega1284
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega1284

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega1284 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega1284P

Contents

- ATmega1284P
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
Configuration

Please use ATmega1284P ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:ATmega1284P]
platform = atmelavr
board = ATmega1284P
```

You can override default ATmega1284P settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega1284P.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:ATmega1284P]
platform = atmelavr
board = ATmega1284P

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

ATmega1284P has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

```plaintext
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
```
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega16

Contents

- ATmega16
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega16 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:ATmega16]
platform = atmelavr
board = ATmega16
```

You can override default ATmega16 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega16.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:ATmega16]
platform = atmelavr
board = ATmega16

; change microcontroller
board_build.mcu = atmega16
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ATmega16 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega162

Contents

- ATmega162
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Configuration

Please use `ATmega162` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:ATmega162]
platform = atmelavr
board = ATmega162
```

You can override default `ATmega162` settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega162.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega162]
platform = atmelavr
board = ATmega162

; change microcontroller
board_build.mcu = atmega162

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support `ATmega162` board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega164A

**Contents**

- ATmega164A
  - Hardware
  - Configuration
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA164A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega164A ID for board option in “platformio.ini” *(Project Configuration File):*

```
[env:ATmega164A]
platform = atmelavr
board = ATmega164A
```

You can override default ATmega164A settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega164A.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega164A]
platform = atmelavr
board = ATmega164A

; change microcontroller
board_build.mcu = atmega164a

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATmega164A board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
ATmega164P/PA

Contents

• ATmega164P/PA
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA164P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega164P ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:ATmega164P]
platform = atmelavr
board = ATmega164P
```

You can override default ATmega164P/PA settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega164P.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:ATmega164P]
platform = atmelavr
board = ATmega164P

; change microcontroller
board_build.mcu = atmega164p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega164P/PA has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega168/A

#### Contents

- **ATmega168/A**
  - *Hardware*
  - *Configuration*
  - *Debugging*
  - *Frameworks*

#### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>
Configuration

Please use ATmega168 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega168]
platform = atmelavr
board = ATmega168
```

You can override default ATmega168/A settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega168.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega168]
platform = atmelavr
board = ATmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

ATmega168/A has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega168P/PA
PlatformIO Documentation, Release 5.0.5a1

Contents

- ATmega168P/PA
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega168P ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega168P]
platform = atmelavr
board = ATmega168P
```

You can override default ATmega168P/PA settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega168P.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega168P]
platform = atmelavr
board = ATmega168P

; change microcontroller
board_build.mcu = atmega168p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

1.12. Boards
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ATmega168P/PA has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega168PB

Contents

- ATmega168PB
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>
## Configuration

Please use ATmega168PB ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATmega168PB]
platform = atmelavr
board = ATmega168PB
```

You can override default ATmega168PB settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega168PB.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega168PB]
platform = atmelavr
board = ATmega168PB

; change microcontroller
board_build.mcu = atmega168pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

`Debugging` currently does not support ATmega168PB board.

## Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## ATmega2560

### Contents

- ATmega2560
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega2560 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:ATmega2560]
platform = atmelavr
board = ATmega2560
```

You can override default ATmega2560 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega2560.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega2560]
platform = atmelavr
board = ATmega2560

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega2560 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

```
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
```
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega2561

Contents

- ATmega2561
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2561</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega2561 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega2561]
platform = atmelavr
board = ATmega2561
```

You can override default ATmega2561 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega2561.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega2561]
platform = atmelavr
board = ATmega2561

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega2561
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging currently does not support ATmega2561 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega32

Contents

- ATmega32
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega32 ID for board option in “platformio.ini” (Project Configuration File):
You can override default ATmega32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:ATmega32]
platform = atmelavr
board = ATmega32

; change microcontroller
board_build.mcu = atmega32

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*Debugging* currently does not support ATmega32 board.

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## ATmega324A

### Contents

- **ATmega324A**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA324A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

### Configuration

Please use ATmega324A ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATmega324A]
platform = atmelavr
board = ATmega324A
```

You can override default ATmega324A settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega324A.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:ATmega324A]
platform = atmelavr
board = ATmega324A

; change microcontroller
board_build.mcu = atmega324a

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

* Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega324A has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega324P

Contents

- ATmega324P
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA324P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega324P ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:ATmega324P]
platform = atmelavr
board = ATmega324P
```

You can override default ATmega324P settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega324P.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:ATmega324P]
platform = atmelavr
board = ATmega324P

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega324p

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega324P has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega324PA

#### Contents

- **ATmega324PA**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
Microcontroller | ATMEGA324PA
---|---
Frequency | 16MHz
Flash | 32KB
RAM | 2KB
Vendor | Microchip

### Configuration

Please use ATmega324PA ID for board option in "platformio.ini" (Project Configuration File):

```
[env:ATmega324PA]
platform = atmelavr
board = ATmega324PA
```

You can override default ATmega324PA settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega324PA.json. For example, board_build.mcu, board_build.

```
[env:ATmega324PA]
platform = atmelavr
board = ATmega324PA

; change microcontroller
board_build.mcu = atmega324pa

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in "platformio.ini" (Project Configuration File).

ATmega324PA has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega324PB

Contents

- ATmega324PB
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market—and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA324PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega324PB ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega324PB]
platform = atmelavr
board = ATmega324PB
```

You can override default ATmega324PB settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega324PB.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega324PB]
platform = atmelavr
board = ATmega324PB

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega324pb

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging currently does not support ATmega324PB board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega328

Contents

- ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega328 ID for board option in “platformio.ini” (Project Configuration File):
You can override default ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega328]
platform = atmelavr
board = ATmega328

; change microcontroller
board_build.mcu = atmega328

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega328 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega328P/PA

- **ATmega328P/PA**
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega328P ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega328P]
platform = atmelavr
board = ATmega328P
```

You can override default ATmega328P/PA settings per build environment using *board_**** option, where *** is a JSON object path from board manifest ATmega328P.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega328P]
platform = atmelavr
board = ATmega328P

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).
ATmega328P/PA has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega328PB

#### Contents

- ATmega328PB
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market—and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

#### Configuration

Please use **ATmega328PB** ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:ATmega328PB]
platform = atmelavr
board = ATmega328PB
```
You can override default ATmega328PB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega328PB.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```plaintext
[env:ATmega328PB]
platform = atmelavr
board = ATmega328PB

; change microcontroller
board_build.mcu = atmega328pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATmega328PB board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATmega48/A**

**Contents**

- ATmega48/A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

**1.12. Boards**
Configuration

Please use ATmega48 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega48]
platform = atmelavr
board = ATmega48
```

You can override default ATmega48/A settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega48.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega48]
platform = atmelavr
board = ATmega48

; change microcontroller
board_build.mcu = atmega48

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

ATmega48/A has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega48P/PA
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA48P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega48P ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:ATmega48P]
platform = atmelavr
board = ATmega48P
```

You can override default ATmega48P/PA settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega48P.json. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:ATmega48P]
platform = atmelavr
board = ATmega48P

; change microcontroller
board_build.mcu = atmega48p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega48P/PA has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATmega48PB**

**Contents**

- **ATmega48PB**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel AVR:** Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMEGA48PB</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
<td>Microchip</td>
</tr>
</tbody>
</table>
Configuration

Please use ATmega48PB ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:ATmega48PB]
platform = atmelavr
board = ATmega48PB
```

You can override default ATmega48PB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega48PB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega48PB]
platform = atmelavr
board = ATmega48PB

; change microcontroller
board_build.mcu = atmega48pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

`Debugging` currently does not support ATmega48PB board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega64/A

Contents

- ATmega64/A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega64 ID for *board* option in “platformio.ini” (*Project Configuration File)*:

```ini
[env:ATmega64]
platform = atmelavr
board = ATmega64
```

You can override default ATmega64/A settings per build environment using *board_*** option, where *** is a JSON object path from board manifest ATmega64.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:ATmega64]
platform = atmelavr
board = ATmega64

; change microcontroller
board_build.mcu = atmega64

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATmega64/A board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega640
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA640</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega640 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:ATmega640]
platform = atmelavr
board = ATmega640
```

You can override default ATmega640 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega640.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega640]
platform = atmelavr
board = ATmega640

; change microcontroller
board_build.mcu = atmega640

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATmega640 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega644/A

Contents

- ATmega644/A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA644A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega644A ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:ATmega644A]
platform = atmelavr
board = ATmega644A
```

You can override default ATmega644/A settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega644A.json. For example, `board_build.mcu,board_build.f_cpu, etc.`

```
[env:ATmega644A]
platform = atmelavr
board = ATmega644A
; change microcontroller
```

(continues on next page)
```python
board_build.mcu = atmega644a
 ; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATmega644/A board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATmega644P/PA**

**Contents**

- ATmega644P/PA
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA644P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

**Configuration**

Please use ATmega644P ID for board option in “platformio.ini” (*Project Configuration File*):
You can override default ATmega644P/PA settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega644P.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega644P]
platform = atmelavr
board = ATmega644P

; change microcontroller
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ATmega644P/PA has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega8/A

**Contents**

- ATmega8/A
  - Hardware
Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega8 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega8]
platform = atmelavr
board = ATmega8
```

You can override default ATmega8/A settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega8]
platform = atmelavr
board = ATmega8

; change microcontroller
board_build.mcu = atmega8

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ATmega8/A has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>simavr</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega8515

**Contents**

- **ATmega8515**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA8515</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

### Configuration

Please use ATmega8515 ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:ATmega8515]
platform = atmelavr
board = ATmega8515
```

You can override default ATmega8515 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega8515.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:ATmega8515]
platform = atmelavr
board = ATmega8515

; change microcontroller
board_build.mcu = atmega8515

; change MCU frequency
board_build.f_cpu = 16000000L

## Debugging

*Debugging* currently does not support ATmega8515 board.

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## ATmega8535

### Contents

- **ATmega8535**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA8535</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>
Configuration

Please use ATmega8535 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:ATmega8535]
platform = atmelavr
board = ATmega8535
```

You can override default ATmega8535 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega8535.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega8535]
platform = atmelavr
board = ATmega8535

; change microcontroller
board_build.mcu = atmega8535

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

`Debugging` currently does not support ATmega8535 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega88/A

Contents

- ATmega88/A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA88</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega88 ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:ATmega88]
platform = atmelavr
board = ATmega88
```

You can override default ATmega88/A settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega88.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega88]
platform = atmelavr
board = ATmega88

; change microcontroller
board_build.mcu = atmega88

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

ATmega88/A has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega88P/PA

Contents

- ATmega88P/PA
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA88P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega88P ID for board option in "platformio.ini" (Project Configuration File):

```
[env:ATmega88P]
platform = atmelavr
board = ATmega88P
```

You can override default ATmega88P/PA settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega88P.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega88P]
platform = atmelavr
board = ATmega88P

; change microcontroller
```

(continues on next page)
board_build.mcu = \texttt{atmega88p}

; change MCU frequency
board_build.f_cpu = \texttt{16000000L}

\section*{Debugging}

\textit{Debugging} - “1-click” solution for debugging with a zero configuration.

\textbf{Warning:} You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging \textit{Tools & Debug Probes} using \texttt{debug_tool} option in “platformio.ini” (Project Configuration File).

ATmega88P/PA has on-board debug probe and \textbf{IS READY} for debugging. You don’t need to use/buy external debug probe.

\begin{center}
\begin{tabular}{|l|c|c|}
\hline
Compatible Tools & On-board & Default \\
\hline
\texttt{simavr} & Yes & Yes \\
\hline
\end{tabular}
\end{center}

\section*{Frameworks}

\begin{center}
\begin{tabular}{|l|p{13cm}|}
\hline
Name & Description \\
\hline
\texttt{Arduino} & Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences \\
\hline
\end{tabular}
\end{center}

\section*{ATmega88PB}

\subsection*{Contents}

- \texttt{ATmega88PB}
  - \texttt{Hardware}
  - \texttt{Configuration}
  - \texttt{Debugging}
  - \texttt{Frameworks}

\section*{Hardware}

Platform \textit{Atmel AVR}: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA88PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

### Configuration

Please use ATmega88PB ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATmega88PB]
platform = atmelavr
board = ATmega88PB
```

You can override default ATmega88PB settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `ATmega88PB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega88PB]
platform = atmelavr
board = ATmega88PB

; change microcontroller
board_build.mcu = atmega88pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support ATmega88PB board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny13

#### Contents

- ATtiny13
  - Hardware
  - Configuration
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones— they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>9MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use **attiny13** ID for *board* option in **“platformio.ini” (Project Configuration File)**:

```ini
[env:attiny13]
platform = atmelavr
board = attiny13
```

You can override default ATtiny13 settings per build environment using **board_*** option, where *** is a JSON object path from board manifest **attiny13.json**. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:attiny13]
platform = atmelavr
board = attiny13

; change microcontroller
board_build.mcu = attiny13

; change MCU frequency
board_build.f_cpu = 9600000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in **“platformio.ini” (Project Configuration File)**.

ATtiny13 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>simavr</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny13A

#### Contents

- ATtiny13A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY13A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>9MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

#### Configuration

Please use **attiny13a** ID for **board** option in “platformio.ini” (Project Configuration File):

```
[env:attiny13a]
platform = atmelavr
board = attiny13a
```

You can override default ATtiny13A settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `attiny13a.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ATtiny13A has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Bluefruit Micro

Contents

- Adafruit Bluefruit Micro
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `bluefruitmicro` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:bluefruitmicro]
platform = atmelavr
board = bluefruitmicro
```

You can override default Adafruit Bluefruit Micro settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bluefruitmicro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:bluefruitmicro]
platform = atmelavr
board = bluefruitmicro

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in "platformio.ini" (Project Configuration File).

Adafruit Bluefruit Micro has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Circuit Playground Classic

Contents

- Adafruit Circuit Playground Classic
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `circuitplay_classic` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:circuitplay_classic]
platform = atmelavr
board = circuitplay_classic
```

You can override default Adafruit Circuit Playground Classic settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `circuitplay_classic.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```
[env:circuitplay_classic]
platform = atmelavr
board = circuitplay_classic

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

Adafruit Circuit Playground Classic has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Feather 328P

**Contents**

- *Adafruit Feather 328P*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
**PlatformIO Documentation, Release 5.0.5a1**

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `feather328p` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:feather328p]
platform = atmelavr
board = feather328p
```

You can override default Adafruit Feather 328P settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `feather328p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:feather328p]
platform = atmelavr
board = feather328p

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Adafruit Feather 328P has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Feather 32u4

Contents

- Adafruit Feather 32u4
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `feather32u4` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:feather32u4]
platform = atmelavr
board = feather32u4
```

You can override default Adafruit Feather 32u4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `feather32u4.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:feather32u4]
platform = atmelavr
board = feather32u4

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Feather 32u4 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Adafruit Flora

#### Contents

- *Adafruit Flora*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

### Configuration

Please use `flora8` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:flora8]
platform = atmelavr
board = flora8
```

You can override default Adafruit Flora settings per build environment using `board_***` option, where *** is a JSON object path from board manifest flora8.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:flora8]
platform = atmelavr
board = flora8

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Flora has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Gemma

Contents

- Adafruit Gemma
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use gemma ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:gemma]
platform = atmelavr
board = gemma
```

You can override default Adafruit Gemma settings per build environment using board_*** option, where *** is a JSON object path from board manifest gemma.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:gemma]
platform = atmelavr
board = gemma

; change microcontroller
```

(continues on next page)
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

Adafruit Gemma has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Adafruit ItsyBitsy 3V/8MHz**

**Contents**

- *Adafruit ItsyBitsy 3V/8MHz*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

### Configuration

Please use itsybitsy32u4_3V ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:itsybitsy32u4_3V]
platform = atmelavr
board = itsybitsy32u4_3V
```

You can override default Adafruit ItsyBitsy 3V/8MHz settings per build environment using `board_***` option, where *** is a JSON object path from board manifest itsybitsy32u4_3V.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:itsybitsy32u4_3V]
platform = atmelavr
board = itsybitsy32u4_3V

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Adafruit ItsyBitsy 3V/8MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit ItsyBitsy 5V/16MHz

Contents

- Adafruit ItsyBitsy 5V/16MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

- | Microcontroller | ATMEGA32U4 |
- | Frequency       | 16MHz      |
- | Flash           | 28KB       |
- | RAM             | 2.50KB     |
- | Vendor          | Adafruit   |

Configuration

Please use itsybitsy32u4_5V ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:itsybitsy32u4_5V]
platform = atmelavr
board = itsybitsy32u4_5V
```

You can override default Adafruit ItsyBitsy 5V/16MHz settings per build environment using board_*** option, where *** is a JSON object path from board manifest itsybitsy32u4_5V.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:itsybitsy32u4_5V]
platform = atmelavr
board = itsybitsy32u4_5V
```

; change microcontroller

(continues on next page)
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

Adafruit ItsyBitsy 5V/16MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Adafruit Metro

**Contents**

- Adafruit Metro
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
**Configuration**

Please use metro ID for board option in “platformio.ini” (Project Configuration File):

```
[env:metro]
platform = atmelavr
board = metro
```

You can override default Adafruit Metro settings per build environment using board_*** option, where *** is a JSON object path from board manifest metro.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:metro]
platform = atmelavr
board = metro

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit Metro has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>avr-stub</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><em>simavr</em></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Pro Trinket 3V/12MHz (FTDI)

Contents

- Adafruit Pro Trinket 3V/12MHz (FTDI)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>12MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `protrinket3ftdi` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:protrinket3ftdi]
platform = atmelavr
board = protrinket3ftdi
```

You can override default Adafruit Pro Trinket 3V/12MHz (FTDI) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `protrinket3ftdi.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:protrinket3ftdi]
platform = atmelavr
board = protrinket3ftdi
; change microcontroller
```
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 12000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit Pro Trinket 3V/12MHz (FTDI) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Pro Trinket 3V/12MHz (USB)

Contents

- Adafruit Pro Trinket 3V/12MHz (USB)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>12MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `protrinket3` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:protrinket3]
platform = atmelavr
board = protrinket3
```

You can override default Adafruit Pro Trinket 3V/12MHz (USB) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `protrinket3.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:protrinket3]
platform = atmelavr
board = protrinket3

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 12000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Pro Trinket 3V/12MHz (USB) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Pro Trinket 5V/16MHz (FTDI)

Contents

- Adafruit Pro Trinket 5V/16MHz (FTDI)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use protrinket5ftdi ID for board option in “platformio.ini” (Project Configuration File):

```
[env:protrinket5ftdi]
platform = atmelavr
board = protrinket5ftdi
```

You can override default Adafruit Pro Trinket 5V/16MHz (FTDI) settings per build environment using board_*** option, where *** is a JSON object path from board manifest protrinket5ftdi.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:protrinket5ftdi]
platform = atmelavr
board = protrinket5ftdi
```

; change microcontroller

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Pro Trinket 5V/16MHz (FTDI) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Adafruit Pro Trinket 5V/16MHz (USB)

#### Contents

- *Adafruit Pro Trinket 5V/16MHz (USB)*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `protrinket5` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:protrinket5]
platform = atmelavr
board = protrinket5
```

You can override default Adafruit Pro Trinket 5V/16MHz (USB) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `protrinket5.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:protrinket5]
platform = atmelavr
board = protrinket5

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

Adafruit Pro Trinket 5V/16MHz (USB) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Trinket 3V/8MHz

Contents

- Adafruit Trinket 3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `trinket3` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:trinket3]
platform = atmelavr
board = trinket3
```

You can override default Adafruit Trinket 3V/8MHz settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `trinket3.json`. For example, `board_build.mcu,board_build.f_cpu,etc`.

```ini
[env:trinket3]
platform = atmelavr
board = trinket3

; change microcontroller
```

(continues on next page)
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 8000000L

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Trinket 3V/8MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Adafruit Trinket 5V/16MHz

#### Contents

- *Adafruit Trinket 5V/16MHz*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
### Configuration

Please use `trinket5` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:trinket5]
platform = atmelavr
board = trinket5
```

You can override default Adafruit Trinket 5V/16MHz settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `trinket5.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:trinket5]
platform = atmelavr
board = trinket5

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Adafruit Trinket 5V/16MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Alorium Hinj

Contents

- Alorium Hinj
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Alorium Technology</td>
</tr>
</tbody>
</table>

Configuration

Please use `alorium_hinj` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:alorium_hinj]
platform = atmelavr
board = alorium_hinj
```

You can override default Alorium Hinj settings per build environment using `board_***` option, where *** is a JSON object path from board manifest alorium_hinj.json. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:alorium_hinj]
platform = atmelavr
board = alorium_hinj

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p
;
change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Alorium Hinj has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Alorium Sno

Contents

- Alorium Sno
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Alorium Technology</td>
</tr>
</tbody>
</table>

**Configuration**

Please use alorium_sno ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:alorium_sno]
platform = atmelavr
board = alorium_sno
```

You can override default Alorium Sno settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `alorium_sno.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:alorium_sno]
platform = atmelavr
board = alorium_sno

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Alorium Sno has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Alorium XLR8

Contents

- Alorium XLR8
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Alorium Technology</td>
</tr>
</tbody>
</table>

Configuration

Please use `alorium_xlr8` ID for *board* option in “`platformio.ini`” *(Project Configuration File)*:

```
[env:alorium_xlr8]
platform = atmelavr
board = alorium_xlr8
```

You can override default Alorium XLR8 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest `alorium_xlr8.json`. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:alorium_xlr8]
platform = atmelavr
board = alorium_xlr8
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

Alorium XLR8 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

**Name** | **Description**
--- | ---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

Altair

**Contents**

- Altair
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones—they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA256RFR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>248KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>makerlab.mx</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `altair` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:altair]
platform = atmelavr
board = altair
```

You can override default Altair settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `altair.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:altair]
platform = atmelavr
board = altair

; change microcontroller
board_build.mcu = atmega256rfr2

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support Altair board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Anarduino MiniWireless**

**Contents**

- Anarduino MiniWireless
  - Hardware
  - Configuration
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Anarduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `miniwireless` ID for `board` option in “*platformio.ini*” *(Project Configuration File)*:

```ini
[env:miniwireless]
platform = atmelavr
board = miniwireless
```

You can override default Anarduino MiniWireless settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `miniwireless.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:miniwireless]
platform = atmelavr
board = miniwireless

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “*platformio.ini*” *(Project Configuration File)*.

Anarduino MiniWireless has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

**Name** | **Description**
--- | ---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

### Arduboy

**Contents**

- Arduboy
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduboy</td>
</tr>
</tbody>
</table>

### Configuration

Please use `arduboy` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:arduboy]
platform = atmelavr
board = arduboy
```

You can override default Arduboy settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `arduboy.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Arduboy has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>simavr</strong></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduboy DevKit

**Contents**

- **Arduboy DevKit**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduboy</td>
</tr>
</tbody>
</table>

Configuration

Please use `arduboy_devkit` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:arduboy_devkit]
platform = atmelavr
board = arduboy_devkit
```

You can override default Arduboy DevKit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `arduboy_devkit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:arduboy_devkit]
platform = atmelavr
board = arduboy_devkit

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Arduboy DevKit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1.12. Boards
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino BT ATmega168

Contents

- Arduino BT ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use btatmega168 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:btatmega168]
platform = atmelavr
board = btatmega168
```

You can override default Arduino BT ATmega168 settings per build environment using board_*** option, where *** is a JSON object path from board manifest btatmega168.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:btatmega168]
platform = atmelavr
board = btatmega168

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino BT ATmega168 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Arduino BT ATmega328

#### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market—and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.
Configuration

Please use `btatmega328` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:btatmega328]
platform = atmelavr
board = btatmega328
```

You can override default Arduino BT ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `btatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:btatmega328]
platform = atmelavr
board = btatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - "1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Arduino BT ATmega328 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

```
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
```
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Duemilanove or Diecimila ATmega168

Contents

- Arduino Duemilanove or Diecimila ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use diecimilaatmega168 ID for board option in "platformio.ini" (Project Configuration File):

```python
[env:diecimilaatmega168]
platform = atmelavr
board = diecimilaatmega168
```

You can override default Arduino Duemilanove or Diecimila ATmega168 settings per build environment using board_*** option, where *** is a JSON object path from board manifest diecimilaatmega168.json. For example, board_build.mcu, board_build.f_cpu, etc.

```python
[env:diecimilaatmega168]
platform = atmelavr
board = diecimilaatmega168
; change microcontroller
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Arduino Duemilanove or Diecimila ATmega168 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Duemilanove or Diecimila ATmega328

Contents

- Arduino Duemilanove or Diecimila ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
## Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

## Configuration

Please use `diecimilaatmega328` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:diecimilaatmega328]
platform = atmelavr
board = diecimilaatmega328
```

You can override default Arduino Duemilanove or Diecimila ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `diecimilaatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:diecimilaatmega328]
platform = atmelavr
board = diecimilaatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Duemilanove or Diecimila ATmega328 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Esplora

Contents

- Arduino Esplora
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use esplora ID for board option in "platformio.ini" (Project Configuration File):

```
[env:esplora]
platform = atmelavr
board = esplora
```

You can override default Arduino Esplora settings per build environment using board_*** option, where *** is a JSON object path from board manifest esplora.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:esplora]
platform = atmelavr
board = esplora
; change microcontroller
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Arduino Esplora has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Ethernet

Contents

- Arduino Ethernet
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
Microcontroller | ATMEGA328P
--- | ---
Frequency | 16MHz
Flash | 31.50KB
RAM | 2KB
Vendor | Arduino

### Configuration

Please use `ethernet` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:ethernet]
platform = atmelavr
board = ethernet
```

You can override default Arduino Ethernet settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ethernet.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ethernet]
platform = atmelavr
board = ethernet

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

Arduino Ethernet has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Fio

Contents

- Arduino Fio
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `fio` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:fio]
platform = atmelavr
board = fio
```

You can override default Arduino Fio settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `fio.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:fio]
platform = atmelavr
board = fio

; change microcontroller
board_build.mcu = atmega328p
```

(continues on next page)
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Arduino Fio has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Industrial 101

Contents

- Arduino Industrial 101
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Configuration

Please use chiwawa ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:chiwawa]
platform = atmelavr
board = chiwawa

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

You can override default Arduino Industrial 101 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chiwawa.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

Arduino Industrial 101 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

```plaintext
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
```
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Leonardo

Contents

• Arduino Leonardo
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use leonardo ID for board option in “platformio.ini” (Project Configuration File):

```
[env:leonardo]
platform = atmelavr
board = leonardo
```

You can override default Arduino Leonardo settings per build environment using board_*** option, where *** is a JSON object path from board manifest leonardo.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:leonardo]
platform = atmelavr
board = leonardo

; change microcontroller
```
(continues on next page)
PlatformIO Documentation, Release 5.0.5a1

(continued from previous page)

\begin{verbatim}
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
\end{verbatim}

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in *“platformio.ini” (Project Configuration File)*.

Arduino Leonardo has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

\[
\begin{array}{|c|c|c|}
\hline
\text{Compatible Tools} & \text{On-board} & \text{Default} \\
\hline
simavr & Yes & Yes \\
\hline
\end{array}
\]

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Arduino Leonardo ETH**

**Contents**

- **Arduino Leonardo ETH**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
### PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

### Configuration

Please use **leonardoeth ID** for **board** option in “platformio.ini” (*Project Configuration File*):

```ini
[env:leonardoeth]
platform = atmelavr
board = leonardoeth
```

You can override default Arduino Leonardo ETH settings per build environment using **board_*** option, where *** is a JSON object path from board manifest `leonardoeth.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:leonardoeth]
platform = atmelavr
board = leonardoeth

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using **debug_tool** option in “platformio.ini” (*Project Configuration File*).

Arduino Leonardo ETH has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino LilyPad ATmega168

Contents

- Arduino LilyPad ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use lilypadatmega168 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:lilypadatmega168]
platform = atmelavr
board = lilypadatmega168
```

You can override default Arduino LilyPad ATmega168 settings per build environment using board_*** option, where *** is a JSON object path from board manifest lilypadatmega168.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:lilypadatmega168]
platform = atmelavr
board = lilypadatmega168
; change microcontroller
```
board_build.mcu = atmega168
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino LilyPad ATmega168 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino LilyPad ATmega328

Contents

- *Arduino LilyPad ATmega328*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
Microcontroller | ATMEGA328P  
--- | ---  
Frequency | 8MHz  
Flash | 30KB  
RAM | 2KB  
Vendor | Arduino

**Configuration**

Please use `lilypadatmega328` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:lilypadatmega328]
platform = atmelavr
board = lilypadatmega328
```

You can override default Arduino LilyPad ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lilypadatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lilypadatmega328]
platform = atmelavr
board = lilypadatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in "platformio.ini" (Project Configuration File).

Arduino LilyPad ATmega328 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino LilyPad USB

Contents

- Arduino LilyPad USB
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market—and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use LilyPadUSB ID for board option in “platformio.ini” (Project Configuration File):

```
[env:LilyPadUSB]
platform = atmelavr
board = LilyPadUSB
```

You can override default Arduino LilyPad USB settings per build environment using board_*** option, where *** is a JSON object path from board manifest LilyPadUSB.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:LilyPadUSB]
platform = atmelavr
board = LilyPadUSB
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in *“platformio.ini” (Project Configuration File)*.

Arduino LilyPad USB has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Arduino Mega ADK**

**Contents**

- Arduino Mega ADK
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
Microcontroller | ATMEGA2560  
---|---
Frequency | 16MHz  
Flash | 248KB  
RAM | 8KB  
Vendor | Arduino

### Configuration

Please use `megaADK` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:megaADK]
platform = atmelavr
board = megaADK
```

You can override default Arduino Mega ADK settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `megaADK.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:megaADK]
platform = atmelavr
board = megaADK

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in "platformio.ini" *(Project Configuration File)*.

Arduino Mega ADK has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>avr-stub</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Mega or Mega 2560 ATmega1280

Contents

- Arduino Mega or Mega 2560 ATmega1280
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1280</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>124KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use megaatmega1280 ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env: megaatmega1280]
platform = atmelavr
board = megaatmega1280
```

You can override default Arduino Mega or Mega 2560 ATmega1280 settings per build environment using board_*** option, where *** is a JSON object path from board manifest megaatmega1280.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env: megaatmega1280]
platform = atmelavr
board = megaatmega1280

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega1280

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

_E Debugging_ - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging _Tools & Debug Probes_ using _debug_tool_ option in “platformio.ini” (Project Configuration File).

Arduino Mega or Mega 2560 ATmega1280 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)

#### Contents

- Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform _Atmel AVR:_ Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>248KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `megaatmega2560` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:megaatmega2560]
platform = atmelavr
board = megaatmega2560
```

You can override default Arduino Mega or Mega 2560 ATmega2560 (Mega 2560) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `megaatmega2560.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:megaatmega2560]
platform = atmelavr
board = megaatmega2560

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Mega or Mega 2560 ATmega2560 (Mega 2560) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

Arduino Micro

Contents

- Arduino Micro
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use micro ID for board option in “platformio.ini” (Project Configuration File):

```
[env:micro]
platform = atmelavr
board = micro
```

You can override default Arduino Micro settings per build environment using board_*** option, where *** is a JSON object path from board manifest micro.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:micro]
platform = atmelavr
board = micro
```

(continues on next page)
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Micro has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Mini ATmega168

**Contents**

- Arduino Mini ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `miniatmega168` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:miniatmega168]
platform = atmelavr
board = miniatmega168
```

You can override default Arduino Mini ATmega168 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `miniatmega168.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:miniatmega168]
platform = atmelavr
board = miniatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Mini ATmega168 has on-board debug probe and *IS READY* for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Mini ATmega328

Contents

- Arduino Mini ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `miniatmega328` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:miniatmega328]
platform = atmelavr
board = miniatmega328
```

You can override default Arduino Mini ATmega328 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `miniatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:miniatmega328]
platform = atmelavr
board = miniatmega328

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Arduino Mini ATmega328 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino NG or older ATmega168

Contents

- Arduino NG or older ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `atmegangatmega168` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:atmegangatmega168]
platform = atmelavr
board = atmegangatmega168
```

You can override default Arduino NG or older ATmega168 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `atmegangatmega168.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:atmegangatmega168]
platform = atmelavr
board = atmegangatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino NG or older ATmega168 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino NG or older ATmega8

Contents

- Arduino NG or older ATmega8
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>7KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `atmegangatmega8` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:atmegangatmega8]
platform = atmelavr
board = atmegangatmega8
```

You can override default Arduino NG or older ATmega8 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `atmegangatmega8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:atmegangatmega8]
platform = atmelavr
board = atmegangatmega8

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega8

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

Arduino NG or older ATmega8 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Nano ATmega168

Contents

- Arduino Nano ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

1.12. Boards
### Configuration

Please use `nanoatmega168` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:nanoatmega168]
platform = atmelavr
board = nanoatmega168
```

You can override default Arduino Nano ATmega168 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nanoatmega168.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nanoatmega168]
platform = atmelavr
board = nanoatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Arduino Nano ATmega168 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Nano ATmega328

Contents

- Arduino Nano ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use nanoatmega328 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:nanoatmega328]
platform = atmelavr
board = nanoatmega328
```

You can override default Arduino Nano ATmega328 settings per build environment using board_*** option, where *** is a JSON object path from board manifest nanoatmega328.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nanoatmega328]
platform = atmelavr
board = nanoatmega328
;
```
(continues on next page)
PlatformIO Documentation, Release 5.0.5a1

board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Arduino Nano ATmega328 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

### Arduino Nano ATmega328 (New Bootloader)

**Contents**

- Arduino Nano ATmega328 (New Bootloader)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `nanoatmega328new` ID for `board` option in *“platformio.ini“ (Project Configuration File)*:

```ini
[env:nanoatmega328new]
platform = atmelavr
board = nanoatmega328new
```

You can override default Arduino Nano ATmega328 (New Bootloader) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nanoatmega328new.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nanoatmega328new]
platform = atmelavr
board = nanoatmega328new

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini“ (Project Configuration File)*.

Arduino Nano ATmega328 (New Bootloader) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>

Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)

Contents

- Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use pro8MHZatmega168 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:pro8MHZatmega168]
platform = atmelavr
board = pro8MHZatmega168
```

You can override default Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz) settings per build environment using board_*** option, where *** is a JSON object path from board manifest pro8MHZatmega168.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:pro8MHZatmega168]
platform = atmelavr
board = pro8MHZatmega168
```

(continues on next page)
; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)**

**Contents**

- *Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `pro16MHzatmega168` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:pro16MHzatmega168]
platform = atmelavr
board = pro16MHzatmega168
```

You can override default Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `pro16MHzatmega168.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pro16MHzatmega168]
platform = atmelavr
board = pro16MHzatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)

Contents

- Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `pro8MHzatmega328` ID for *board* option in "platformio.ini" (Project Configuration File):

```ini
[env:pro8MHzatmega328]
platform = atmelavr
board = pro8MHzatmega328
```

You can override default Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `pro8MHzatmega328.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:pro8MHzatmega328]
platform = atmelavr
board = pro8MHzatmega328

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L

Debugger

Debugger - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)

Contents

- *Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `pro16MHzatmega328` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:pro16MHzatmega328]
platform = atmelavr
board = pro16MHzatmega328
```

You can override default Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `pro16MHzatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:pro16MHzatmega328]
platform = atmelavr
board = pro16MHzatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz) has on-board debug probe and is *READY* for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Robot Control

Contents

- Arduino Robot Control
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use robotControl ID for board option in “platformio.ini” (Project Configuration File):

```
[env:robotControl]
platform = atmelavr
board = robotControl
```

You can override default Arduino Robot Control settings per build environment using board_*** option, where *** is a JSON object path from board manifest robotControl.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:robotControl]
platform = atmelavr
board = robotControl

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* - "1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Arduino Robot Control has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Robot Motor

Contents

- Arduino Robot Motor
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `robotMotor` ID for `board` option in “PLATFORMIO.INI” *(Project Configuration File)*:

```
[env:robotMotor]
platform = atmelavr
board = robotMotor
```

You can override default Arduino Robot Motor settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `robotMotor.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:robotMotor]
platform = atmelavr
board = robotMotor

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “PLATFORMIO.INI” *(Project Configuration File)*.

Arduino Robot Motor has on-board debug probe and is READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Uno

Contents

- Arduino Uno
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones— they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use uno ID for board option in "platformio.ini" (Project Configuration File):

```
[env:uno]
platform = atmelavr
board = uno
```

You can override default Arduino Uno settings per build environment using board_*** option, where *** is a JSON object path from board manifest uno.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:uno]
platform = atmelavr
board = uno

; change microcontroller
board_build.mcu = atmega328p
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in *“platformio.ini” (Project Configuration File)*.

Arduino Uno has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

Arduino Yun

Contents

- **Arduino Yun**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `yun` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:yun]
platform = atmelavr
board = yun
```

You can override default Arduino Yun settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `yun.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:yun]
platform = atmelavr
board = yun

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Yun has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Yun Mini

Contents

- Arduino Yun Mini
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `yunmini` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:yunmini]
  platform = atmelavr
  board = yunmini
```

You can override default Arduino Yun Mini settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `yunmini.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:yunmini]
  platform = atmelavr
  board = yunmini

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

Arduino Yun Mini has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Atmel AT90PWM216**

**Contents**

- *Atmel AT90PWM216*
  - Hardware
  - Configuration
  - Debugging

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
## Configuration

Please use `at90pwm216` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:at90pwm216]
platform = atmelavr
board = at90pwm216
```

You can override default Atmel AT90PWM216 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `at90pwm216.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:at90pwm216]
platform = atmelavr
board = at90pwm216

; change microcontroller
board_build.mcu = at90pwm216

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*Debugging* currently does not support Atmel AT90PWM216 board.

### Atmel AT90PWM316

#### Contents

- *Atmel AT90PWM316*
  - Hardware
  - Configuration
  - Debugging

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
**PlatformIO Documentation, Release 5.0.5a1**

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT90PWM316</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

### Configuration

Please use `at90pwm316` ID for `board` option in “platformio.ini” (*Project Configuration File*):

```ini
[env:at90pwm316]
platform = atmelavr
board = at90pwm316
```

You can override default Atmel AT90PWM316 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `at90pwm316.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:at90pwm316]
platform = atmelavr
board = at90pwm316

; change microcontroller
board_build.mcu = at90pwm316

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support Atmel AT90PWM316 board.

### BQ ZUM BT-328

**Contents**

- **BQ ZUM BT-328**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
Microcontroller | ATMEGA328P  
---|---  
Frequency | 16MHz  
Flash | 28KB  
RAM | 2KB  
Vendor | BQ

**Configuration**

Please use `zumbt328` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:zumbt328]
platform = atmelavr
board = zumbt328
```

You can override default BQ ZUM BT-328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `zumbt328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:zumbt328]
platform = atmelavr
board = zumbt328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

BQ ZUM BT-328 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>avr-stub</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

876 Chapter 1. Contents
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

BitWizard Raspduino

Contents

- BitWizard Raspduino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BitWizard</td>
</tr>
</tbody>
</table>

Configuration

Please use **raspuino ID** for **board** option in “platformio.ini” *(Project Configuration File)*:

```
[env:raspuino]
platform = atmelavr
board = raspuino
```

You can override default BitWizard Raspduino settings per build environment using **board_*** option, where *** is a JSON object path from board manifest raspuino.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:raspuino]
platform = atmelavr
board = raspuino

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

BitWizard Raspduino has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Controllino Maxi**

**Contents**

- **Controllino Maxi**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>248KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Controllino</td>
</tr>
</tbody>
</table>

### Configuration

Please use `controllino_maxi` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:controllino_maxi]
platform = atmelavr
board = controllino_maxi
```

You can override default Controllino Maxi settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `controllino_maxi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:controllino_maxi]
platform = atmelavr
board = controllino_maxi

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Controllino Maxi has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Controllino Maxi Automation

Contents

- Controllino Maxi Automation
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>248KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Controllino</td>
</tr>
</tbody>
</table>

Configuration

Please use `controllino_maxi_automation` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:controllino_maxi_automation]
platform = atmelavr
board = controllino_maxi_automation
```

You can override default Controllino Maxi Automation settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `controllino_maxi_automation.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:controllino_maxi_automation]
platform = atmelavr
board = controllino_maxi_automation
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Controllino Maxi Automation has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Controllino Mega

Contents

- Controllino Mega
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>248KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Controllino</td>
</tr>
</tbody>
</table>

Configuration

Please use controllino_mega ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:controllino_mega]
platform = atmelavr
board = controllino_mega
```

You can override default Controllino Mega settings per build environment using board_*** option, where *** is a JSON object path from board manifest controllino_mega.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:controllino_mega]
platform = atmelavr
board = controllino_mega

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Controllino Mega has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Controllino Mini

Contents

- Controllino Mini
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Controllino</td>
</tr>
</tbody>
</table>

Configuration

Please use `controllino_mini` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:controllino_mini]
platform = atmelavr
board = controllino_mini
```

You can override default Controllino Mini settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `controllino_mini.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:controllino_mini]
platform = atmelavr
board = controllino_mini

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Controllino Mini has on-board debug probe and is ready for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digispark Pro

Contents

- Digispark Pro
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY167</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digistump</td>
</tr>
</tbody>
</table>

### Configuration

Please use `digispark-pro` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:digispark-pro]
platform = atmelavr
board = digispark-pro
```

You can override default Digispark Pro settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `digispark-pro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:digispark-pro]
platform = atmelavr
board = digispark-pro

; change microcontroller
board_build.mcu = attiny167

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support Digispark Pro board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Digispark Pro (16 MHz) (64 byte buffer)

- **Contents**
  - Digispark Pro (16 MHz) (64 byte buffer)
    - Hardware
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY167</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digistump</td>
</tr>
</tbody>
</table>

Configuration

Please use **digispark-pro64** ID for **board** option in “platformio.ini” (**Project Configuration File**):

```ini
[env:digispark-pro64]
platform = atmelavr
board = digispark-pro64
```

You can override default Digispark Pro (16 MHz) (64 byte buffer) settings per build environment using **board_*** option, where *** is a JSON object path from board manifest digispark-pro64.json. For example, **board_build.mcu**, **board_build.f_cpu**, etc.

```ini
[env:digispark-pro64]
platform = atmelavr
board = digispark-pro64

; change microcontroller
board_build.mcu = attiny167

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**Debugging** currently does not support Digispark Pro (16 MHz) (64 byte buffer) board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Digispark Pro (32 byte buffer)

Contents

- Digispark Pro (32 byte buffer)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY167</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>14.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digistump</td>
</tr>
</tbody>
</table>

Configuration

Please use digispark-pro32 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:digispark-pro32]
platform = atmelavr
board = digispark-pro32
```

You can override default Digispark Pro (32 byte buffer) settings per build environment using board_*** option, where *** is a JSON object path from board manifest digispark-pro32.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:digispark-pro32]
platform = atmelavr
board = digispark-pro32

; change microcontroller
board_build.mcu = attiny167

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging currently does not support Digispark Pro (32 byte buffer) board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digispark USB

Contents

- Digispark USB
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>5.87KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digistump</td>
</tr>
</tbody>
</table>

Configuration

Please use digispark-tiny ID for board option in "platformio.ini" (Project Configuration File):

```
[env:digispark-tiny]
platform = atmelavr
board = digispark-tiny
```

You can override default Digispark USB settings per build environment using board_*** option, where *** is a JSON object path from board manifest digispark-tiny.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:digispark-tiny]
platform = atmelavr
board = digispark-tiny

; change microcontroller
```

(continues on next page)
board_build.mcu = attiny85
; change MCU frequency
board_build.f_cpu = 16500000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Digispark USB has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Dwenguino

Contents

- Dwenguino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT90USB646</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>60KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Dwengo</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `dwenguino` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:dwenguino]
platform = atmelavr
board = dwenguino
```

You can override default Dwenguino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `dwenguino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:dwenguino]
platform = atmelavr
board = dwenguino

; change microcontroller
board_build.mcu = at90usb646

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support Dwenguino board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Elektor Uno R4**

**Contents**

- *Elektor Uno R4*
  - Hardware
  - Configuration
  - Debugging
Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Elektor</td>
</tr>
</tbody>
</table>

Configuration

Please use `elektor_uno_r4` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:elektor_uno_r4]
platform = atmelavr
board = elektor_uno_r4
```

You can override default Elektor Uno R4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `elektor_uno_r4.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:elektor_uno_r4]
platform = atmelavr
board = elektor_uno_r4

; change microcontroller
board_build.mcu = atmega328pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging currently does not support Elektor Uno R4 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Engduino 3

Contents

- Engduino 3
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Engduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `engduinov3` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:engduinov3]
platform = atmelavr
board = engduinov3
```

You can override default Engduino 3 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `engduinov3.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:engduinov3]
platform = atmelavr
board = engduinov3

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Engduino 3 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

EnviroDIY Mayfly

Contents

- EnviroDIY Mayfly
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>EnviroDIY</td>
</tr>
</tbody>
</table>
Configuration

Please use `mayfly` ID for `board` option in "platformio.ini" (Project Configuration File):

```python
[env:mayfly]
platform = atmelavr
board = mayfly
```

You can override default EnviroDIY Mayfly settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mayfly.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:mayfly]
platform = atmelavr
board = mayfly

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

_Debbuging_ - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging _Tools & Debug Probes_ using `debug_tool` option in "platformio.ini" (Project Configuration File).

EnviroDIY Mayfly has on-board debug probe and _IS READY_ for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

FYSETC F6 V1.3
Contents

- FYSETC F6 V1.3
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>252KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>FYSETC</td>
</tr>
</tbody>
</table>

Configuration

Please use `fysetc_f6_13` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:fysetc_f6_13]
platform = atmelavr
board = fysetc_f6_13
```

You can override default FYSETC F6 V1.3 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `fysetc_f6_13.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:fysetc_f6_13]
platform = atmelavr
board = fysetc_f6_13

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

FYSETC F6 V1.3 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny1634

Contents

- Generic ATtiny1634
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny1634</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>
Configuration

Please use attiny1634 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:attiny1634]
platform = atmelavr
board = attiny1634
```

You can override default Generic ATtiny1634 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny1634.json. For example, board_build.mcu, board_build. f_cpu, etc.

```
[env:attiny1634]
platform = atmelavr
board = attiny1634

; change microcontroller
board_build.mcu = attiny1634

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

Debugging currently does not support Generic ATtiny1634 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny167

Contents

- Generic ATtiny167
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
PlatformIO Documentation, Release 5.0.5a1

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY167</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `attiny167` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:attiny167]
platform = atmelavr
board = attiny167
```

You can override default Generic ATtiny167 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `attiny167.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:attiny167]
platform = atmelavr
board = attiny167

; change microcontroller
board_build.mcu = attiny167

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* currently does not support Generic ATtiny167 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny2313
Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY2313</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use attiny2313 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:attiny2313]
platform = atmelavr
board = attiny2313
```

You can override default Generic ATtiny2313 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny2313.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:attiny2313]
platform = atmelavr
board = attiny2313

; change microcontroller
board_build.mcu = attiny2313

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Generic ATtiny2313 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Arduino</em></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Generic ATtiny24

#### Contents

- *Generic ATtiny24*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>
Configuration

Please use attiny24 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:attiny24]
platform = atmelavr
board = attiny24
```

You can override default Generic ATtiny24 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny24.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:attiny24]
platform = atmelavr
board = attiny24

; change microcontroller
board_build.mcu = attiny24

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Generic ATtiny24 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny25
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `attiny25` ID for `board` option in “platformio.ini” *(Project Configuration File):*

```
[env:attiny25]
platform = atmelavr
board = attiny25
```

You can override default Generic ATtiny25 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `attiny25.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny25]
platform = atmelavr
board = attiny25

; change microcontroller
board_build.mcu = attiny25

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Generic ATtiny25 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Generic ATtiny261

#### Contents

- *Generic ATtiny261*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY261</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>
Configuration

Please use `attiny261` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:attiny261]
platform = atmelavr
board = attiny261
```

You can override default Generic ATtiny261 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny261.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny261]
platform = atmelavr
board = attiny261

; change microcontroller
board_build.mcu = attiny261

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

`Debugging` currently does not support Generic ATtiny261 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny4313

Contents

- `Generic ATtiny4313`
  - `Hardware`
  - `Configuration`
  - `Debugging`
  - `Frameworks`
Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY4313</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use attiny4313 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:attiny4313]
platform = atmelavr
board = attiny4313
```

You can override default Generic ATtiny4313 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny4313.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:attiny4313]
platform = atmelavr
board = attiny4313

; change microcontroller
board_build.mcu = attiny4313

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Generic ATtiny4313 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny43U

Contents

- Generic ATtiny43U
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY43U</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use attiny43 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:attiny43]
platform = atmelavr
board = attiny43
```

You can override default Generic ATtiny43U settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny43.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:attiny43]
platform = atmelavr
board = attiny43

; change microcontroller
```

(continues on next page)
Debugging

Debugging currently does not support Generic ATtiny43U board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny44

Contents

- Generic ATtiny44
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY44</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use attiny44 ID for board option in “platformio.ini” (Project Configuration File):
You can override default Generic ATtiny44 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny44.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:attiny44]
platform = atmelavr
board = attiny44

; change microcontroller
board_build.mcu = attiny44

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*).

Generic ATtiny44 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Generic ATtiny441

**Contents**

- *Generic ATtiny44*
  - Hardware
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY441</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `attiny441` ID for *board* option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:attiny441]
platform = atmelavr
board = attiny441
```

You can override default Generic ATtiny441 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `attiny441.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:attiny441]
platform = atmelavr
board = attiny441

; change microcontroller
board_build.mcu = attiny441

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*Debugging* currently does not support Generic ATtiny441 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Generic ATtiny45

Contents

- Generic ATtiny45
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `attiny45` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:attiny45]
platform = atmelavr
board = attiny45
```

You can override default Generic ATtiny45 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny45.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:attiny45]
platform = atmelavr
board = attiny45

; change microcontroller
board_build.mcu = attiny45

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Generic ATtiny45 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny461

Contents

- Generic ATtiny461
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY461</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>
Configuration

Please use attiny461 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:attiny461]
platform = atmelavr
board = attiny461
```

You can override default Generic ATtiny461 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny461.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:attiny461]
platform = atmelavr
board = attiny461

; change microcontroller
board_build.mcu = attiny461

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

Debugging currently does not support Generic ATtiny461 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny48

Contents

- Generic ATtiny48
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use attiny48 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:attiny48]
platform = atmelavr
board = attiny48
```

You can override default Generic ATtiny48 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny48.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:attiny48]
platform = atmelavr
board = attiny48

; change microcontroller
board_build.mcu = attiny48

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

Debugging currently does not support Generic ATtiny48 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny828
PlatformIO Documentation, Release 5.0.5a1

Contents

- Generic ATtiny828
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY828</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use attiny828 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:attiny828]
platform = atmelavr
board = attiny828
```

You can override default Generic ATtiny828 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny828.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:attiny828]
platform = atmelavr
board = attiny828

; change microcontroller
board_build.mcu = attiny828

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

Debugging currently does not support Generic ATtiny828 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny84

Contents

- Generic ATtiny84
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY84</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use attiny84 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:attiny84]
platform = atmelavr
board = attiny84
```

You can override default Generic ATtiny84 settings per build environment using board_*** option, where *** is a JSON object path from board manifest attiny84.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:attiny84]
platform = atmelavr
board = attiny84

; change microcontroller
```

(continues on next page)
board_build.mcu = attiny84

; change MCU frequency
board_build.f_cpu = 8000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Generic ATtiny84 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Generic ATtiny841**

**Contents**

- *Generic ATtiny841*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Configuration

Please use `attiny841` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:attiny841]
platform = atmelavr
board = attiny841
```

You can override default Generic ATtiny841 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny841.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:attiny841]
platform = atmelavr
board = attiny841

; change microcontroller
board_build.mcu = attiny841

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*Debugging* currently does not support Generic ATtiny841 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Generic ATtiny85

#### Contents

- Generic ATtiny85
  - Hardware
  - Configuration

1.12. Boards
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `attiny85` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:attiny85]
platform = atmelavr
board = attiny85
```

You can override default Generic ATtiny85 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `attiny85.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny85]
platform = atmelavr
board = attiny85

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

Generic ATtiny85 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Generic ATtiny861

#### Contents

- Generic ATtiny861
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY861</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `attiny861` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:attiny861]
platform = atmelavr
board = attiny861
```

You can override default Generic ATtiny861 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `attiny861.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:attiny861]
platform = atmelavr
board = attiny861

; change microcontroller
board_build.mcu = attiny861

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging currently does not support Generic ATtiny861 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny87

Contents

- Generic ATtiny87
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY87</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>
Configuration

Please use `attiny87` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:attiny87]
platform = atmelavr
board = attiny87
```

You can override default Generic ATtiny87 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny87.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:attiny87]
platform = atmelavr
board = attiny87

; change microcontroller
board_build.mcu = attiny87

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* currently does not support Generic ATtiny87 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Generic ATtiny88

Contents

- Generic ATtiny88
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY88</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `attiny88` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:attiny88]
platform = atmelavr
board = attiny88
```

You can override default Generic ATtiny88 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny88.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:attiny88]
platform = atmelavr
board = attiny88

; change microcontroller
board_build.mcu = attiny88

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**Debugging** currently does not support Generic ATtiny88 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**LightBlue Bean**
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Punch Through</td>
</tr>
</tbody>
</table>

Configuration

Please use `lightblue-bean` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:lightblue-bean]
platform = atmelavr
board = lightblue-bean
```

You can override default LightBlue Bean settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lightblue-bean.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lightblue-bean]
platform = atmelavr
board = lightblue-bean

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

LightBlue Bean has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

LightBlue Bean+

Contents

- LightBlue Bean+
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Punch Through</td>
</tr>
</tbody>
</table>
Configuration

Please use `lightblue-beanplus` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:lightblue-beanplus]
platform = atmelavr
board = lightblue-beanplus
```

You can override default LightBlue Bean+ settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lightblue-beanplus.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lightblue-beanplus]
platform = atmelavr
board = lightblue-beanplus

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in `platformio.ini` (Project Configuration File).

LightBlue Bean+ has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

LightUp
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LightUp</td>
</tr>
</tbody>
</table>

Configuration

Please use *lightup* ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```python
[env:lightup]
platform = atmelavr
board = lightup
```

You can override default LightUp settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lightup.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:lightup]
platform = atmelavr
board = lightup

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

LightUp has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Linino One

#### Contents

- *Linino One*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Linino</td>
</tr>
</tbody>
</table>

#### Configuration

Please use one ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:one]
platform = atmelavr
board = one
```
You can override default Linino One settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `one.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:one]
platform = atmelavr
board = one

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

Linino One has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## LinkIt Smart 7688 Duo

**Contents**

- **LinkIt Smart 7688 Duo**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MediaTek Labs</td>
</tr>
</tbody>
</table>

Configuration

Please use `smart7688` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:smart7688]
platform = atmelavr
board = smart7688
```

You can override default LinkIt Smart 7688 Duo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `smart7688.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:smart7688]
platform = atmelavr
board = smart7688

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

LinkIt Smart 7688 Duo has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

LoRa32u4II (868-915MHz)

Contents

- LoRa32u4II (868-915MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BSFrance</td>
</tr>
</tbody>
</table>

Configuration

Please use lora32u4II ID for board option in "platformio.ini" (Project Configuration File):

```plaintext
[env:lora32u4II]
platform = atmelavr
board = lora32u4II
```

You can override default LoRa32u4II (868-915MHz) settings per build environment using board_*** option, where *** is a JSON object path from board manifest lora32u4II.json. For example, board_build.mcu, board_build.f_cpu, etc.

```plaintext
[env:lora32u4II]
platform = atmelavr
board = lora32u4II

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4
;

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

LoRa32u4II (868-915MHz) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

LowPowerLab MightyHat

Contents

- LowPowerLab MightyHat
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LowPowerLab</td>
</tr>
</tbody>
</table>

**Configuration**

Please use mightyhat ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:mightyhat]
platform = atmelavr
board = mightyhat
```

You can override default LowPowerLab MightyHat settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `mightyhat.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mightyhat]
platform = atmelavr
board = mightyhat

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

LowPowerLab MightyHat has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

LowPowerLab Moteino

Contents

- **LowPowerLab Moteino**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LowPowerLab</td>
</tr>
</tbody>
</table>

Configuration

Please use moteino ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:moteino]
platform = atmelavr
board = moteino
```

You can override default LowPowerLab Moteino settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `moteino.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.:

```
[env:moteino]
platform = atmelavr
board = moteino

; change microcontroller
```
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

LowPowerLab Moteino has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

LowPowerLab Moteino (8Mhz)

**Contents**

- LowPowerLab Moteino (8Mhz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LowPowerLab</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `moteino8mhz` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:moteino8mhz]
platform = atmelavr
board = moteino8mhz
```

You can override default LowPowerLab Moteino (8Mhz) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `moteino8mhz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:moteino8mhz]
platform = atmelavr
board = moteino8mhz

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

LowPowerLab Moteino (8Mhz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## LowPowerLab MoteinoMEGA

### Contents

- LowPowerLab MoteinoMEGA
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LowPowerLab</td>
</tr>
</tbody>
</table>

### Configuration

Please use `moteinomega ID` for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:moteinomega]
platform = atmelavr
board = moteinomega
```

You can override default LowPowerLab MoteinoMEGA settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `moteinomega.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:moteinomega]
platform = atmelavr
board = moteinomega

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

LowPowerLab MoteinoMEGA has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Microduino Core (Atmega168PA@16M,5V)

Contents

- Microduino Core (Atmega168PA@16M,5V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
Microcontroller | ATMEGA168P  
---|---  
Frequency | 16MHz  
Flash | 15.50KB  
RAM | 1KB  
Vendor | Microduino

**Configuration**

Please use `168pa16m` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:168pa16m]
platform = atmelavr
board = 168pa16m
```

You can override default Microduino Core (Atmega168PA@16M,5V) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `168pa16m.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```
[env:168pa16m]
platform = atmelavr
board = 168pa16m

; change microcontroller
board_build.mcu = atmega168p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

Microduino Core (Atmega168PA@16M,5V) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Microduino Core (Atmega168PA@8M,3.3V)

Contents

- Microduino Core (Atmega168PA@8M,3.3V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA168P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>15.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

Configuration

Please use 168pa8m ID for *board* option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:168pa8m]
platform = atmelavr
board = 168pa8m
```

You can override default Microduino Core (Atmega168PA@8M,3.3V) settings per build environment using *board_*** option, where *** is a JSON object path from board manifest 168pa8m.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:168pa8m]
platform = atmelavr
board = 168pa8m
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega168p

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Microduino Core (Atmega168PA@8M,3.3V) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Microduino Core (Atmega328P@16M,5V)

Contents

- *Microduino Core (Atmega328P@16M,5V)*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.
**Configuration**

Please use 328p16m ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:328p16m]
platform = atmelavr
board = 328p16m
```

You can override default Microduino Core (Atmega328P@16M,5V) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `328p16m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:328p16m]
platform = atmelavr
board = 328p16m

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Microduino Core (Atmega328P@16M,5V) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

```
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
```
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Microduino Core (Atmega328P@8M,3.3V)

Contents

- Microduino Core (Atmega328P@8M,3.3V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

Configuration

Please use 328p8m ID for *board* option in “`platformio.ini` (Project Configuration File):`

```
[env:328p8m]
platform = atmelavr
board = 328p8m
```

You can override default Microduino Core (Atmega328P@8M,3.3V) settings per build environment using *board_*** option, where *** is a JSON object path from board manifest 328p8m.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:328p8m]
platform = atmelavr
board = 328p8m

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 8000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

Microduino Core *(Atmega328P@8M,3.3V)* has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Microduino Core USB (ATmega32U4@16M,5V)**

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones - they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use 32u416m ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:32u416m]
platform = atmelavr
board = 32u416m
```

You can override default Microduino Core USB (ATmega32U4@16M,5V) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest 32u416m.json. For example, `board_build.mcu,board_build.f_cpu`, etc.

```
[env:32u416m]
platform = atmelavr
board = 32u416m

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Microduino Core USB (ATmega32U4@16M,5V) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Microduino Core+ (ATmega1284P@16M,5V)**

**Contents**

- Microduino Core+ (ATmega1284P@16M,5V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use 1284p16m ID for board option in “platformio.ini” (Project Configuration File):

```
[env:1284p16m]
platform = atmelavr
board = 1284p16m
```

You can override default Microduino Core+ (ATmega1284P@16M,5V) settings per build environment using board_*** option, where *** is a JSON object path from board manifest 1284p16m.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:1284p16m]
platform = atmelavr
board = 1284p16m
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Microduino Core+ (ATmega1284P@16M,5V) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Microduino Core+ (ATmega1284P@8M,3.3V)

Contents

- Microduino Core+ (ATmega1284P@8M,3.3V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

### Configuration

Please use `1284p8m` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:1284p8m]
platform = atmelavr
board = 1284p8m
```

You can override default Microduino Core+ (ATmega1284P@8M,3.3V) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `1284p8m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:1284p8m]
platform = atmelavr
board = 1284p8m

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in "platformio.ini" (Project Configuration File).

Microduino Core+ (ATmega1284P@8M,3.3V) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

```
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
```
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Microduino Core+ (Atmega644PA@16M,5V)

Contents

- Microduino Core+ (Atmega644PA@16M,5V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA644P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>63KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

Configuration

Please use 644pa16m ID for *board* option in “platformio.ini” (Project Configuration File):

```ini
[env:644pa16m]
platform = atmelavr
board = 644pa16m
```

You can override default Microduino Core+ (Atmega644PA@16M,5V) settings per build environment using board_*** option, where *** is a JSON object path from board manifest 644pa16m.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:644pa16m]
platform = atmelavr
board = 644pa16m

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega644p

; change MCU_frequency
board_build.f_cpu = 16000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Microduino Core+ (Atmega644PA@16M,5V) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Microduino Core+ (Atmega644PA@8M,3.3V)

Contents

- Microduino Core+ (Atmega644PA@8M,3.3V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming
### Configuration

Please use `644pa8m` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:644pa8m]
platform = atmelavr
board = 644pa8m
```

You can override default Microduino Core+ (Atmega644PA@8M,3.3V) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `644pa8m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:644pa8m]
platform = atmelavr
board = 644pa8m

; change microcontroller
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Microduino Core+ (Atmega644PA@8M,3.3V) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

OpenEnergyMonitor emonPi

Contents

- OpenEnergyMonitor emonPi
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>OpenEnergyMonitor</td>
</tr>
</tbody>
</table>

Configuration

Please use `emonpi` ID for `board` option in “platformio.ini” (Project Configuration File):

```plaintext
[env:emonpi]
platform = atmelavr
board = emonpi
```

You can override default OpenEnergyMonitor emonPi settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `emonpi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:emonpi]
platform = atmelavr
board = emonpi
```

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

OpenEnergyMonitor emonPi has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Original Prusa i3 MK3 Multi Material 2.0 Upgrade**

**Contents**

- *Original Prusa i3 MK3 Multi Material 2.0 Upgrade*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Prusa 3D</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `prusa_mm_control` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:prusa_mm_control]
platform = atmelavr
board = prusa_mm_control
```

You can override default Original Prusa i3 MK3 Multi Material 2.0 Upgrade settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `prusa_mm_control.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:prusa_mm_control]
platform = atmelavr
board = prusa_mm_control

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in "platformio.ini" (Project Configuration File).

Original Prusa i3 MK3 Multi Material 2.0 Upgrade has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

PanStamp AVR

Contents

- PanStamp AVR
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PanStamp</td>
</tr>
</tbody>
</table>

Configuration

Please use `panStampAVR` ID for board option in “platformio.ini” (Project Configuration File):

```
[env:panStampAVR]
platform = atmelavr
board = panStampAVR
```

You can override default PanStamp AVR settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `panStampAVR.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:panStampAVR]
platform = atmelavr
board = panStampAVR
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

PanStamp AVR has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Pinoccio Scout**

**Contents**

- Pinoccio Scout
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA256RFR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>248KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Pinoccio</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `pinoccio` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:pinoccio]
platform = atmelavr
board = pinoccio
```

You can override default Pinoccio Scout settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `pinoccio.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pinoccio]
platform = atmelavr
board = pinoccio

; change microcontroller
board_build.mcu = atmega256rfr2

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support Pinoccio Scout board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Pololu A-Star 32U4**

**Contents**

- *Pololu A-Star 32U4*
  - Hardware
Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Pololu Corporation</td>
</tr>
</tbody>
</table>

Configuration

Please use `a-star32U4` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:a-star32U4]
platform = atmelavr
board = a-star32U4
```

You can override default Pololu A-Star 32U4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `a-star32U4.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:a-star32U4]
platform = atmelavr
board = a-star32U4

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
Pololu A-Star 32U4 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Prusa RAMBo

**Contents**

- *Prusa RAMBo*
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

### Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>252KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Prusa 3D</td>
</tr>
</tbody>
</table>

### Configuration

Please use `prusa_rambo ID` for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:prusa_rambo]
platform = atmelavr
board = prusa_rambo
```
You can override default Prusa RAMBo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `prusa_rambo.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:prusa_rambo]
platform = atmelavr
board = prusa_rambo

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Prusa RAMBo has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Quirkbot

**Contents**

- **Quirkbot**
  - Hardware
  - Configuration
  - Debugging

1.12. Boards 959
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Quirkbot</td>
</tr>
</tbody>
</table>

Configuration

Please use *quirkbot* ID for *board* option in “platformio.ini” (Project Configuration File):

```
[env:quirkbot]
platform = atmelavr
board = quirkbot
```

You can override default Quirkbot settings per build environment using *board_*** option, where *** is a JSON object path from board manifest *quirkbot.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:quirkbot]
platform = atmelavr
board = quirkbot

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Quirkbot has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

RedBearLab Blend

Contents

- RedBearLab Blend
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RedBearLab</td>
</tr>
</tbody>
</table>

Configuration

Please use blend ID for board option in "platformio.ini" (Project Configuration File):

```
[env:blend]
platform = atmelavr
board = blend
```

You can override default RedBearLab Blend settings per build environment using board_*** option, where *** is a JSON object path from board manifest blend.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:blend]
platform = atmelavr
board = blend
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

RedBearLab Blend has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

RedBearLab Blend Micro 3.3V/16MHz (overclock)

Contents

- RedBearLab Blend Micro 3.3V/16MHz (overclock)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market—and easily adapt to new ones—they are based on the industry’s most code-efficient architecture for C and assembly programming.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RedBearLab</td>
</tr>
</tbody>
</table>

## Configuration

Please use `blendmicro16` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:blendmicro16]
platform = atmelavr
board = blendmicro16
```

You can override default RedBearLab Blend Micro 3.3V/16MHz (overclock) settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `blendmicro16.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:blendmicro16]
platform = atmelavr
board = blendmicro16

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

RedBearLab Blend Micro 3.3V/16MHz (overclock) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

RedBearLab Blend Micro 3.3V/8MHz

Contents

- RedBearLab Blend Micro 3.3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RedBearLab</td>
</tr>
</tbody>
</table>

Configuration

Please use blendmicro8 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:blendmicro8]
platform = atmelavr
board = blendmicro8
```

You can override default RedBearLab Blend Micro 3.3V/8MHz settings per build environment using board_option, where *** is a JSON object path from board manifest blendmicro8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:blendmicro8]
platform = atmelavr
board = blendmicro8
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

RedBearLab Blend Micro 3.3V/8MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

RepRap RAMBo

**Contents**

- *RepRap RAMBo*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>252KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RepRap</td>
</tr>
</tbody>
</table>

### Configuration

Please use reprap_rambo ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:reprap_rambo]
platform = atmelavr
board = reprap_rambo
```

You can override default RepRap RAMBo settings per build environment using board_*** option, where *** is a JSON object path from board manifest reprap_rambo.json. For example, board_build.mcu, board_build. f_cpu, etc.

```ini
[env:reprap_rambo]
platform = atmelavr
board = reprap_rambo

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

RepRap RAMBo has on-board debug probe and is READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ GaLoRa

Contents

- **SODAQ GaLoRa**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use `sodaq_galora` ID for *board* option in *“platformio.ini” (Project Configuration File)*:

```ini
[env:sodaq_galora]
platform = atmelavr
board = sodaq_galora
```

You can override default SODAQ GaLoRa settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sodaq_galora.json`. For example, `board_build.mcu, board_build.f_cpu, etc.

```ini
[env:sodaq_galora]
platform = atmelavr
board = sodaq_galora
```

; change microcontroller

(continues on next page)
board_build.mcu = atmega1284p
;
change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

SODAQ GaLoRa has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ Mbili

Contents

- SODAQ Mbili
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming
Configuration

Please use `sodaq_mbili` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:sodaq_mbili]
platform = atmelavr
board = sodaq_mbili
```

You can override default SODAQ Mbili settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `sodaq_mbili.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sodaq_mbili]
platform = atmelavr
board = sodaq_mbili

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SODAQ Mbili has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ Moja

Contents

- SODAQ Moja
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use sodaq_moja ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sodaq_moja]
platform = atmelavr
board = sodaq_moja
```

You can override default SODAQ Moja settings per build environment using board_*** option, where *** is a JSON object path from board manifest sodaq_moja.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sodaq_moja]
platform = atmelavr
board = sodaq_moja
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SODAQ Moja has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ Ndogo

Contents

- *SODAQ Ndogo*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use sodaq_ndogo ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sodaq_ndogo]
platform = atmelavr
board = sodaq_ndogo
```

You can override default SODAQ Ndogo settings per build environment using board_*** option, where *** is a JSON object path from board manifest sodaq_ndogo.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sodaq_ndogo]
platform = atmelavr
board = sodaq_ndogo

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

SODAQ Ndogo has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ Tatu

Contents

- SODAQ Tatu
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use sodaq_tatu ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sodaq_tatu]
platform = atmelavr
board = sodaq_tatu
```

You can override default SODAQ Tatu settings per build environment using board_*** option, where *** is a JSON object path from board manifest sodaq_tatu.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sodaq_tatu]
platform = atmelavr
board = sodaq_tatu

; change microcontroller
board_build.mcu = atmega1284p
```

(continues on next page)
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

SODAQ Tatu has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Sanguino ATmega1284p (16MHz)

Contents

- `Sanguino ATmega1284p (16MHz)`
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sanguino</td>
</tr>
</tbody>
</table>

### Configuration

Please use `sanguino_atmega1284p` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:sanguino_atmega1284p]
platform = atmelavr
board = sanguino_atmega1284p
```

You can override default Sanguino ATmega1284p (16MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sanguino_atmega1284p.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:sanguino_atmega1284p]
platform = atmelavr
board = sanguino_atmega1284p

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sanguino ATmega1284p (16MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Sanguino ATmega1284p (8MHz)**

**Contents**

- **Sanguino ATmega1284p (8MHz)**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sanguino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sanguino_atmega1284_8m` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:sanguino_atmega1284_8m]
platform = atmelavr
board = sanguino_atmega1284_8m
```

You can override default Sanguino ATmega1284p (8MHz) settings per build environment using `board_***` option, where ``` is a JSON object path from board manifest `sanguino_atmega1284_8m.json`. For example, `board_build.mcu, board_build.f_cpu, etc.

```
[env:sanguino_atmega1284_8m]
platform = atmelavr
board = sanguino_atmega1284_8m
; change microcontroller
```
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

Sanguino ATmega1284p (8MHz) has on-board debug probe and is **READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Sanguino ATmega644 or ATmega644A (16 MHz)**

**Contents**

- *Sanguino ATmega644 or ATmega644A (16 MHz)*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

1.12. Boards
### Configuration

Please use `sanguino_atmega644` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:sanguino_atmega644]
platform = atmelavr
board = sanguino_atmega644
```

You can override default Sanguino ATmega644 or ATmega644A (16 MHz) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sanguino_atmega644.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sanguino_atmega644]
platform = atmelavr
board = sanguino_atmega644

; change microcontroller
board_build.mcu = atmega644

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" *(Project Configuration File)*.

Sanguino ATmega644 or ATmega644A (16 MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

```
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
```
Frameworks

### Name | Description
--- | ---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

Sanguino ATmega644 or ATmega644A (8 MHz)

**Contents**

- **Sanguino ATmega644 or ATmega644A (8 MHz)**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA644</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>63KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sanguino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sanguino_atmega644_8m` ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```ini
[leno:sanguino_atmega644_8m]
platform = atmelavr
board = sanguino_atmega644_8m
```

You can override default Sanguino ATmega644 or ATmega644A (8 MHz) settings per build environment using *board_**** option, where *** is a JSON object path from board manifest sanguino_atmega644_8m.json. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```ini
[leno:sanguino_atmega644_8m]
platform = atmelavr
board = sanguino_atmega644_8m

; change microcontroller
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Sanguino ATmega644 or ATmega644A (8 MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Sanguino ATmega644P or ATmega644PA (16 MHz)

Contents

- Sanguino ATmega644P or ATmega644PA (16 MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR:** Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA644P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>63KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sanguino</td>
</tr>
</tbody>
</table>

### Configuration

Please use `sanguino_atmega644p` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:sanguino_atmega644p]
platform = atmelavr
board = sanguino_atmega644p
```

You can override default Sanguino ATmega644P or ATmega644PA (16 MHz) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sanguino_atmega644p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sanguino_atmega644p]
platform = atmelavr
board = sanguino_atmega644p

; change microcontroller
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sanguino ATmega644P or ATmega644PA (16 MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Sanguino ATmega644P or ATmega644PA (8 MHz)

Contents

- Sanguino ATmega644P or ATmega644PA (8 MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA644P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>63KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sanguino</td>
</tr>
</tbody>
</table>

Configuration

Please use sanguino_atmega644p_8m ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sanguino_atmega644p_8m]
platform = atmelavr
board = sanguino_atmega644p_8m
```

You can override default Sanguino ATmega644P or ATmega644PA (8 MHz) settings per build environment using board_*** option, where *** is a JSON object path from board manifest sanguino_atmega644p_8m.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sanguino_atmega644p_8m]
platform = atmelavr
board = sanguino_atmega644p_8m

; change microcontroller
```
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sanguino ATmega644P or ATmega644PA (8 MHz) has on-board debug probe and is ready for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Seeeduino

Contents

- Seeeduino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Configuration

Please use `seeeduino` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:seeeduino]
platform = atmelavr
board = seeeduino
```

You can override default Seeeduino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest seeeduino.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:seeeduino]
platform = atmelavr
board = seeeduino

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Seeeduino has on-board debug probe and *IS READY* for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

SparkFun ATmega128RFA1 Dev Board

Contents

- SparkFun ATmega128RFA1 Dev Board
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA128RFA1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>124KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_satmega128rfal ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:sparkfun_satmega128rfal]
platform = atmelavr
board = sparkfun_satmega128rfal
```

You can override default SparkFun ATmega128RFA1 Dev Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_satmega128rfal.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```ini
[env:sparkfun_satmega128rfal]
platform = atmelavr
board = sparkfun_satmega128rfal
```

(continues on next page)
; change microcontroller
board_build.mcu = atmega128rfa1

; change MCU frequency
board_build.f_cpu = 16000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

SparkFun ATmega128RFA1 Dev Board has on-board debug probe and **is ready** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**SparkFun Digital Sandbox**

**Contents**

- *SparkFun Digital Sandbox*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sparkfun_digitalsandbox` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:sparkfun_digitalsandbox]
platform = atmelavr
board = sparkfun_digitalsandbox

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

SparkFun Digital Sandbox has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Fio V3 3.3V/8MHz

Contents

- SparkFun Fio V3 3.3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_fiov3 ID for board option in "platformio.ini" (Project Configuration File):

```python
[env:sparkfun_fiov3]
platform = atmelavr
board = sparkfun_fiov3
```

You can override default SparkFun Fio V3 3.3V/8MHz settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_fiov3.json. For example, board_build.mcu, board_build.f_cpu, etc.

```python
[env:sparkfun_fiov3]
platform = atmelavr
board = sparkfun_fiov3

; change microcontroller
```

(continues on next page)
PlatformIO Documentation, Release 5.0.5a1

board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

SparkFun Fio V3 3.3V/8MHz has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Makey Makey

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

1.12. Boards
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `sparkfun_makeymakey` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:sparkfun_makeymakey]
platform = atmelavr
board = sparkfun_makeymakey
```

You can override default SparkFun Makey Makey settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sparkfun_makeymakey.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sparkfun_makeymakey]
platform = atmelavr
board = sparkfun_makeymakey

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

#### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun Makey Makey has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Mega Pro 3.3V/8MHz

Contents

- SparkFun Mega Pro 3.3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>252KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_megapro8MHz ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_megapro8MHz]
platform = atmelavr
board = sparkfun_megapro8MHz
```

You can override default SparkFun Mega Pro 3.3V/8MHz settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_megapro8MHz.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sparkfun_megapro8MHz]
platform = atmelavr
board = sparkfun_megapro8MHz

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

SparkFun Mega Pro 3.3V/8MHz has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Mega Pro 5V/16MHz

Contents

- SparkFun Mega Pro 5V/16MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR:* Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>248KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sparkfun_megapro16MHz` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:sparkfun_megapro16MHz]
platform = atmelavr
board = sparkfun_megapro16MHz
```

You can override default SparkFun Mega Pro 5V/16MHz settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_megapro16MHz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sparkfun_megapro16MHz]
platform = atmelavr
board = sparkfun_megapro16MHz

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

SparkFun Mega Pro 5V/16MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>avr-stub</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Mega Pro Mini 3.3V

Contents

- SparkFun Mega Pro Mini 3.3V
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA2560</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>252KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use *sparkfun_megamini* ID for *board* option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_megamini]
platform = atmelavr
board = sparkfun_megamini
```

You can override default SparkFun Mega Pro Mini 3.3V settings per build environment using *board_*** option, where *** is a JSON object path from board manifest *sparkfun_megamini.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:sparkfun_megamini]
platform = atmelavr
board = sparkfun_megamini

; change microcontroller
```

(continues on next page)
board_build.mcu = `atmega2560`

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun Mega Pro Mini 3.3V has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>avr-stub</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun MicroView

**Contents**

- *SparkFun MicroView*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `uview` ID for `board` option in “`platformio.ini` (Project Configuration File)”: 

```ini
[env:uview]
platform = atmelavr
board = uview
```

You can override default SparkFun MicroView settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `uview.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:uview]
platform = atmelavr
board = uview

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

SparkFun MicroView has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Pro Micro 3.3V/8MHz

Contents

- SparkFun Pro Micro 3.3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_promicro8 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:sparkfun_promicro8]
platform = atmelavr
board = sparkfun_promicro8
```

You can override default SparkFun Pro Micro 3.3V/8MHz settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_promicro8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sparkfun_promicro8]
platform = atmelavr
board = sparkfun_promicro8

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L

### Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

SparkFun Pro Micro 3.3V/8MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

**Name** | **Description**
---|---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

### SparkFun Pro Micro 5V/16MHz

**Contents**

- *SparkFun Pro Micro 5V/16MHz*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Configuration

Please use `sparkfun_promicro16` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:sparkfun_promicro16]
platform = atmelavr
board = sparkfun_promicro16
```

You can override default SparkFun Pro Micro 5V/16MHz settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sparkfun_promicro16.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sparkfun_promicro16]
platform = atmelavr
board = sparkfun_promicro16

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun Pro Micro 5V/16MHz has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Qduino Mini

Contents

- SparkFun Qduino Mini
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use `sparkfun_qduinomini` ID for `board` option in “platformio.ini” (Project Configuration File):

```plaintext
[env:sparkfun_qduinomini]
platform = atmelavr
board = sparkfun_qduinomini
```

You can override default SparkFun Qduino Mini settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_qduinomini.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:sparkfun_qduinomini]
platform = atmelavr
board = sparkfun_qduinomini

; change microcontroller
```

(continues on next page)
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (Project Configuration File).

SparkFun Qduino Mini has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun RedBoard

Contents

- *SparkFun RedBoard*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sparkfun_redboard` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:sparkfun_redboard]
platform = atmelavr
board = sparkfun_redboard
```

You can override default SparkFun RedBoard settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sparkfun_redboard.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sparkfun_redboard]
platform = atmelavr
board = sparkfun_redboard

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

SparkFun RedBoard has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Serial 7-Segment Display

Contents

- SparkFun Serial 7-Segment Display
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_serial7seg ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_serial7seg]
platform = atmelavr
board = sparkfun_serial7seg
```

You can override default SparkFun Serial 7-Segment Display settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_serial7seg.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sparkfun_serial7seg]
platform = atmelavr
board = sparkfun_serial7seg

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L

## Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun Serial 7-Segment Display has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>avr-stub</code></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><code>simavr</code></td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

**Name** | **Description**
---|---
**Arduino** | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

## SpellFoundry Sleepy Pi 2

### Contents

- **SpellFoundry Sleepy Pi 2**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SpellFoundry</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sleepypi` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:sleepypi]
platform = atmelavr
board = sleepypi
```

You can override default SpellFoundry Sleepy Pi 2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sleepypi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sleepypi]
platform = atmelavr
board = sleepypi

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SpellFoundry Sleepy Pi 2 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>


Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Talk2 Whisper Node

Contents

- Talk2 Whisper Node
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Wisen</td>
</tr>
</tbody>
</table>

Configuration

Please use `whispernode ID` for `board` option in “platformio.ini” (Project Configuration File):

```
[env:whispernode]
platform = atmelavr
board = whispernode
```

You can override default Talk2 Whisper Node settings per build environment using `board_*** option, where *** is a JSON object path from board manifest `whispernode.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:whispernode]
platform = atmelavr
board = whispernode

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

`; change MCU frequency
board_build.f_cpu = 16000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

Talk2 Whisper Node has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**The Things Uno**

**Contents**

- The Things Uno
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>28KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>The Things Network</td>
</tr>
</tbody>
</table>

### Configuration

Please use the **the_things_uno** ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:the_things_uno]
platform = atmelavr
board = the_things_uno
```

You can override default The Things Uno settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `the_things_uno.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:the_things_uno]
platform = atmelavr
board = the_things_uno

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

The Things Uno has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

TinyCircuits TinyDuino Processor Board

Contents

- TinyCircuits TinyDuino Processor Board
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TinyCircuits</td>
</tr>
</tbody>
</table>

Configuration

Please use tinyduino ID for board option in “platformio.ini” (Project Configuration File):

```
[env:tinyduino]
platform = atmelavr
board = tinyduino
```

You can override default TinyCircuits TinyDuino Processor Board settings per build environment using board_*** option, where *** is a JSON object path from board manifest tinyduino.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:tinyduino]
platform = atmelavr
board = tinyduino
```

; change microcontroller

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

TinyCircuits TinyDuino Processor Board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**TinyCircuits TinyLily Mini Processor**

**Contents**

- *TinyCircuits TinyLily Mini Processor*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s
most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>30KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TinyCircuits</td>
</tr>
</tbody>
</table>

**Configuration**

Please use tinylily ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:tinylily]
platform = atmelavr
board = tinylily
```

You can override default TinyCircuits TinyLily Mini Processor settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `tinylily.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:tinylily]
platform = atmelavr
board = tinylily

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

TinyCircuits TinyLily Mini Processor has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

USBasp stick

Contents

- USBasp stick
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>12MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `usbasp` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:usbasp]
platform = atmelavr
board = usbasp
```

You can override default USBasp stick settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `usbasp.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:usbasp]
platform = atmelavr
board = usbasp

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega8

; change MCU frequency
board_build.f_cpu = 12000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

⚠️ Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

USBasp stick has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Wicked Device WildFire V2

Contents

- Wicked Device WildFire V2
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
Microcontroller | ATMEGA1284P
---|---
Frequency | 16MHz
Flash | 120.00KB
RAM | 16KB
Vendor | Wicked Device

**Configuration**

Please use `wildfirev2` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:wildfirev2]
platform = atmelavr
board = wildfirev2
```

You can override default Wicked Device WildFire V2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `wildfirev2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:wildfirev2]
platform = atmelavr
board = wildfirev2

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Wicked Device WildFire V2 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Wicked Device WildFire V3

Contents

- *Wicked Device WildFire V3*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Wicked Device</td>
</tr>
</tbody>
</table>

Configuration

Please use `wildfirev3` ID for `board` option in “platformio.ini” *(Project Configuration File):*

```ini
[env:wildfirev3]
platform = atmelavr
board = wildfirev3
```

You can override default Wicked Device WildFire V3 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `wildfirev3.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:wildfirev3]
platform = atmelavr
board = wildfirev3

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Wicked Device WildFire V3 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

Name | Description
---|---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

**ftDuino**

**Contents**

- ftDuino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.
### Configuration

Please use ftduino ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:ftduino]
platform = atmelavr
board = ftduino
```

You can override default ftDuino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ftduino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ftduino]
platform = atmelavr
board = ftduino

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

ftDuino has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
nicai-systems BOB3 coding bot

Contents

• nicai-systems BOB3 coding bot
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA88</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>nicai-systems</td>
</tr>
</tbody>
</table>

Configuration

Please use bob3 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:bob3]
platform = atmelavr
board = bob3
```

You can override default nicai-systems BOB3 coding bot settings per build environment using board_*** option, where *** is a JSON object path from board manifest bob3.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:bob3]
platform = atmelavr
board = bob3

; change microcontroller
board_build.mcu = atmega88

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

nicai-systems BOB3 coding bot has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### nicai-systems NIBO 2 robot

#### Contents

- nicai-systems NIBO 2 robot
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **Atmel AVR**: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>nicai-systems</td>
</tr>
</tbody>
</table>
Configuration

Please use nibo2 ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:nibo2]
platform = atmelavr
board = nibo2
```

You can override default nicai-systems NIBO 2 robot settings per build environment using `board_***` option, where *** is a JSON object path from board manifest nibo2.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nibo2]
platform = atmelavr
board = nibo2

; change microcontroller
board_build.mcu = atmega128

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in "platformio.ini" (Project Configuration File).

nicai-systems NIBO 2 robot has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

**Name** | **Description**
---|---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

nicai-systems NIBO burger robot
Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>15MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>nicai-systems</td>
</tr>
</tbody>
</table>

Configuration

Please use niboburger ID for board option in “platformio.ini” (Project Configuration File):

```
[env:niboburger]
platform = atmelavr
board = niboburger
```

You can override default nicai-systems NIBO burger robot settings per build environment using board_*** option, where *** is a JSON object path from board manifest niboburger.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:niboburger]
platform = atmelavr
board = niboburger

; change microcontroller
board_build.mcu = atmega16

; change MCU frequency
board_build.f_cpu = 15000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

nicai-systems NIBO burger robot has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

nicai-systems NIBO burger robot with Tuning Kit

Contents

- nicai-systems NIBO burger robot with Tuning Kit
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>20MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>nicai-systems</td>
</tr>
</tbody>
</table>
Configuration

Please use `niboburger_1284` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:niboburger_1284]
platform = atmelavr
board = niboburger_1284
```

You can override default nicai-systems NIBO burger robot with Tuning Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `niboburger_1284.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:niboburger_1284]
platform = atmelavr
board = niboburger_1284

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 20000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

nicai-systems NIBO burger robot with Tuning Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

nicai-systems NIBObee robot
Hardware

Platform *Atmel AVR*: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>15MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>nicai-systems</td>
</tr>
</tbody>
</table>

Configuration

Please use *nibobee* ID for *board* option in “platformio.ini” (*Project Configuration File*):

```
[env:nibobee]
platform = atmelavr
board = nibobee
```

You can override default nicai-systems NIBObee robot settings per build environment using *board_*** option, where *** is a JSON object path from board manifest *nibobee.json*. For example, *board_build.mcu, board_build.f_cpu*, etc.

```
[env:nibobee]
platform = atmelavr
board = nibobee

; change microcontroller
board_build.mcu = atmega16

; change MCU frequency
board_build.f_cpu = 15000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

nicai-systems NIBObee robot has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

nicai-systems NIBObee robot with Tuning Kit

Contents

- nicai-systems NIBObee robot with Tuning Kit
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1284P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>20MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>nicai-systems</td>
</tr>
</tbody>
</table>
Configuration

Please use nibobee_1284 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nibobee_1284]
platform = atmelavr
board = nibobee_1284
```

You can override default nicai-systems NIBObee robot with Tuning Kit settings per build environment using board_*** option, where *** is a JSON object path from board manifest nibobee_1284.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nibobee_1284]
platform = atmelavr
board = nibobee_1284

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 20000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

nicai-systems NIBObee robot with Tuning Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>simavr</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ubIQio Ardhat
Contents

- ubIQio Ardhat
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel AVR: Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry's most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA328P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ubIQio</td>
</tr>
</tbody>
</table>

Configuration

Please use ardhat ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ardhat]
platform = atmelavr
board = ardhat
```

You can override default ubIQio Ardhat settings per build environment using board_*** option, where *** is a JSON object path from board manifest ardhat.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ardhat]
platform = atmelavr
board = ardhat

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ubIQio Ardhat has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>avr-stub</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>simavr</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### 1.12.4 Atmel megaAVR

**ATmega1608**

**Contents**

- ATmega1608
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1608</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>
Configuration

Please use ATmega1608 ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:ATmega1608]
platform = atmelmegaavr
board = ATmega1608
```

You can override default ATmega1608 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATmega1608.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega1608]
platform = atmelmegaavr
board = ATmega1608

; change microcontroller
board_build.mcu = atmega1608

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATmega1608 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega1609

Contents

- ATmega1609
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA1609</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega1609 ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:ATmega1609]
platform = atmelmegaavr
board = ATmega1609
```

You can override default ATmega1609 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `ATmega1609.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega1609]
platform = atmelmegaavr
board = ATmega1609

; change microcontroller
board_build.mcu = atmega1609

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATmega1609 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega3208
Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA3208</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega3208 ID for board option in **“platformio.ini”** **(Project Configuration File):**

```ini
[env:ATmega3208]
platform = atmelmegaavr
board = ATmega3208
```

You can override default ATmega3208 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `ATmega3208.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATmega3208]
platform = atmelmegaavr
board = ATmega3208

; change microcontroller
board_build.mcu = atmega3208

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**Debugging** currently does not support ATmega3208 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATmega3209

Contents

- ATmega3209
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA3209</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega3209 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:ATmega3209]
platform = atmelmegaavr
board = ATmega3209
```

You can override default ATmega3209 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega3209.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega3209]
platform = atmelmegaavr
board = ATmega3209

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega3209

; change MCU frequency
board_build.f_cpu = 16000000L

**Debugging**

*Debugging* currently does not support ATmega3209 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATmega4808**

### Contents

- **ATmega4808**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

**Hardware**

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA4808</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>48KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

**Configuration**

Please use ATmega4808 ID for **board** option in “platformio.ini” *(Project Configuration File)*:
You can override default ATmega4808 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega4808.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:ATmega4808]
platform = atmelmegaavr
board = ATmega4808

; change microcontroller
board_build.mcu = {atmega4808

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATmega4808 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATmega4809**

**Contents**

- ATmega4809
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.
### Configuration

Please use ATmega4809 ID for `board` option in "platformio.ini" (Project Configuration File):

```python
[env:ATmega4809]
platform = atmelmegaavr
board = ATmega4809
```

You can override default ATmega4809 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega4809.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:ATmega4809]
platform = atmelmegaavr
board = ATmega4809

; change microcontroller
board_build.mcu = atmega4809

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support ATmega4809 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATmega808

**Contents**

- *ATmega808*
  - Hardware
  - Configuration
Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA808</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega808 ID for *board* option in “platformio.ini” (*Project Configuration File*):

```
[env:ATmega808]
platform = atmelmegaavr
board = ATmega808
```

You can override default ATmega808 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest ATmega808.json. For example, *board_build.mcu, board_build.f_cpu*, etc.

```
[env:ATmega808]
platform = atmelmegaavr
board = ATmega808

; change microcontroller
board_build.mcu = atmega808

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATmega808 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
ATmega809

Contents

- ATmega809
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA809</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega809 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega809]
platform = atmelmegaavr
board = ATmega809
```

You can override default ATmega809 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATmega809.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATmega809]
platform = atmelmegaavr
board = ATmega809

; change microcontroller
board_build.mcu = atmega809

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging currently does not support ATmega809 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATtiny1604

Contents

- ATtiny1604
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny1604</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny1604 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATtiny1604]
platform = atmelmegaavr
board = ATtiny1604
```

You can override default ATtiny1604 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATtiny1604.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATtiny1604]
platform = atmelmegaavr
board = ATtiny1604
; change microcontroller
```

(continues on next page)
board_build.mcu = attiny1604

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* currently does not support ATtiny1604 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Arduino</em></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATtiny1606

Contents

- ATtiny1606
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny1606</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny1606 ID for *board* option in “platformio.ini” (*Project Configuration File)*:
You can override default ATtiny1606 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATtiny1606.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:ATtiny1606]
platform = atmelmegaavr
board = ATtiny1606

; change microcontroller
board.build.mcu = attiny1606

; change MCU frequency
board.build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATtiny1606 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATtiny1607**

**Contents**

- **ATtiny1607**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.
**Configuration**

Please use ATtiny1607 ID for `board` option in “`platformio.ini` (Project Configuration File):`

```ini
[env:ATtiny1607]
platform = atmelmegaavr
board = ATtiny1607
```

You can override default ATtiny1607 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny1607.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATtiny1607]
platform = atmelmegaavr
board = ATtiny1607

; change microcontroller
board_build.mcu = attiny1607

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATtiny1607 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATtiny1614**

**Contents**

- ATtiny1614
  - Hardware
  - Configuration
Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY1614</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny1614 ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:ATTiny1614]
platform = atmelmegaavr
board = ATTiny1614
```

You can override default ATtiny1614 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `ATTiny1614.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATTiny1614]
platform = atmelmegaavr
board = ATTiny1614

; change microcontroller
board_build.mcu = attiny1614

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATtiny1614 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
ATtiny1616

Contents

- ATtiny1616
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny1616</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny1616 ID for *board* option in "platformio.ini" (Project Configuration File):

```ini
[env:ATtiny1616]
platform = atmelmegaavr
board = ATtiny1616
```

You can override default ATtiny1616 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny1616.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATtiny1616]
platform = atmelmegaavr
board = ATtiny1616

; change microcontroller
board_build.mcu = attiny1616

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATtiny1616 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATtiny1617

Contents

- ATtiny1617
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny1617</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use **ATtiny1617** ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```plaintext
[env:ATTiny1617]
platform = atmelmegaavr
board = ATtiny1617
```

You can override default ATtiny1617 settings per build environment using *board_*** option, where *** is a JSON object path from board manifest **ATTiny1617.json**. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```plaintext
[env:ATTiny1617]
platform = atmelmegaavr
board = ATtiny1617
; change microcontroller
```
board_build.mcu = attiny1617
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging currently does not support ATtiny1617 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATtiny202

Contents

- ATtiny202
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY202</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny202 ID for board option in “platformio.ini” (Project Configuration File):
You can override default ATtiny202 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATtiny202.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:ATtiny202]
platform = atmelmegaavr
board = ATtiny202

; change microcontroller
board_build.mcu = attiny202

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support ATtiny202 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny204

#### Contents

- `ATtiny204`
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.
### Configuration

Please use ATtiny204 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATtiny204]
platform = atmelmegaavr
board = ATtiny204
```

You can override default ATtiny204 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATtiny204.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATtiny204]
platform = atmelmegaavr
board = ATtiny204

; change microcontroller
board_build.mcu = attiny204

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

**Debugging** currently does not support ATtiny204 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny212

**Contents**

- **ATtiny212**
  - Hardware
  - Configuration
  - Debugging
PlatformIO Documentation, Release 5.0.5a1

- **Frameworks**

**Hardware**

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny212</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

**Configuration**

Please use ATtiny212 ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:ATtiny212]
platform = atmelmegaavr
board = ATtiny212
```

You can override default ATtiny212 settings per build environment using *board_*** option, where *** is a JSON object path from board manifest ATtiny212.json. For example, *board_build.mcu, board_build.f_cpu*, etc.

```ini
[env:ATtiny212]
platform = atmelmegaavr
board = ATtiny212

; change microcontroller
board_build.mcu = attiny212

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATtiny212 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
ATtiny214

Contents

- ATtiny214
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATtiny214</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny214 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:ATtiny214]
platform = atmelmegaavr
board = ATtiny214
```

You can override default ATtiny214 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny214.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATtiny214]
platform = atmelmegaavr
board = ATtiny214

; change microcontroller
board_build.mcu = attiny214

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging currently does not support ATtiny214 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATtiny3216

Contents

- ATtiny3216
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY3216</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny3216 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATTiny3216]
platform = atmelmegaavr
board = ATtiny3216
```

You can override default ATtiny3216 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATtiny3216.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATTiny3216]
platform = atmelmegaavr
board = ATtiny3216
;
; change microcontroller
```

(continues on next page)
board_build.mcu = `attiny3216`

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

**Debugging** currently does not support ATtiny3216 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny3217

#### Contents

- ATtiny3217
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTINY3217</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

#### Configuration

Please use ATtiny3217 ID for `board` option in “platformio.ini” *(Project Configuration File)*:
You can override default ATtiny3217 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATTiny3217.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

### Debugging

* Debugging currently does not support ATtiny3217 board.*

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny402

#### Contents

- ATtiny402
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.
### Configuration

Please use ATtiny402 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATtiny402]
platform = atmelmegaavr
board = ATtiny402
```

You can override default ATtiny402 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest ATtiny402.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATtiny402]
platform = atmelmegaavr
board = ATtiny402

; change microcontroller
board_build.mcu = attiny402

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support ATtiny402 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny404

**Contents**

- *ATtiny404*
  - Hardware
  - Configuration
  - Debugging
Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY404</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny404 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATTiny404]
platform = atmelmegaavr
board = ATTiny404
```

You can override default ATtiny404 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny404.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATTiny404]
platform = atmelmegaavr
board = ATTiny404

; change microcontroller
board_build.mcu = attiny404

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATtiny404 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
ATtiny406

Contents

• ATtiny406
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY406</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny406 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATtiny406]
platform = atmelmegaavr
board = ATtiny406
```

You can override default ATtiny406 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATtiny406.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATtiny406]
platform = atmelmegaavr
board = ATtiny406

; change microcontroller
board_build.mcu = attiny406

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging currently does not support ATtiny406 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATtiny412**

**Contents**

- **ATtiny412**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY412</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

**Configuration**

Please use ATtiny412 ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:ATtiny412]
platform = atmelmegaavr
board = ATtiny412
```

You can override default ATtiny412 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest ATtiny412.json. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:ATtiny412]
platform = atmelmegaavr
board = ATtiny412

; change microcontroller
board_build.mcu = attiny412
```

(continues on next page)
Debugging

Debugging currently does not support ATtiny412 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATtiny414

Contents

- ATtiny414
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY414</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny414 ID for board option in “platformio.ini” (Project Configuration File):
You can override default ATtiny414 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATtiny414.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```{env:ATtiny414}
platform = atmelmegaavr
board = ATtiny414

; change microcontroller
board_build.mcu = attiny414

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATtiny414 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATtiny416**

**Contents**

- ATtiny416
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY416</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

**Configuration**

Please use ATtiny416 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATtiny416]
platform = atmelmegaavr
board = ATtiny416
```

You can override default ATtiny416 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny416.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATtiny416]
platform = atmelmegaavr
board = ATtiny416

; change microcontroller
board_build.mcu = attiny416

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*Debugging* currently does not support ATtiny416 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**ATtiny417**

**Contents**

- **ATtiny417**
  - Hardware
  - Configuration
  - Debugging
Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY417</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATTiny417 ID for board option in “platformio.ini” *(Project Configuration File)*:

```
[env:ATTiny417]
platform = atmelmegaavr
board = ATTiny417
```

You can override default ATTiny417 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATTiny417.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATTiny417]
platform = atmelmegaavr
board = ATTiny417

; change microcontroller
board_build.mcu = attiny417

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATTiny417 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
ATtiny804

Contents

- ATtiny804
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY804</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny804 ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:ATtiny804]
platform = atmelmegaavr
board = ATtiny804
```

You can override default ATtiny804 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny804.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATtiny804]
platform = atmelmegaavr
board = ATtiny804

; change microcontroller
board_build.mcu = attiny804

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATtiny804 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

ATtiny806

Contents

- **ATtiny806**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY806</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny806 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATtiny806]
platform = atmelmegaavr
board = ATtiny806
```

You can override default ATtiny806 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny806.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATtiny806]
platform = atmelmegaavr
board = ATtiny806

; change microcontroller
board_build.mcu = attiny806
```

(continues on next page)
PlatformIO Documentation, Release 5.0.5a1

(continued from previous page)

```c
; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

_Debugging_ currently does not support ATtiny806 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny807

#### Contents

- ATtiny807
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform _Atmel megaAVR_: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY807</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

#### Configuration

Please use ATtiny807 ID for _board_ option in "platformio.ini" (Project Configuration File):
You can override default ATtiny807 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest ATtiny807.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:ATtiny807]
platform = atmelmegaavr
board = ATtiny807

; change microcontroller
board_build.mcu = attiny807

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

Debugging currently does not support ATtiny807 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny814

#### Contents

- ATtiny814
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.
### Configuration

Please use ATtiny814 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ATtiny814]
platform = atmelmegaavr
board = ATtiny814
```

You can override default ATtiny814 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ATtiny814.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ATtiny814]
platform = atmelmegaavr
board = ATtiny814

; change microcontroller
board_build.mcu = attiny814

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* currently does not support ATtiny814 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ATtiny816

#### Contents

- **ATtiny816**
  - Hardware
  - Configuration
  - Debugging
Hardware

Platform *Atmel megaAVR*: 8-bit MCUs built for real-time control with core-independent peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY816</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny816 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATTiny816]
platform = atmelmegaavr
board = ATtiny816
```

You can override default ATtiny816 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATtiny816.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATTiny816]
platform = atmelmegaavr
board = ATtiny816

; change microcontroller
board_build.mcu = attiny816

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support ATtiny816 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
ATtiny817

Contents

- ATtiny817
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Atmel megaAVR: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY817</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATtiny817 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATtiny817]
platform = atmelmegaavr
board = ATtiny817
```

You can override default ATtiny817 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ATtiny817.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ATtiny817]
platform = atmelmegaavr
board = ATtiny817

; change microcontroller
do_build.mcu = attiny817

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging currently does not support ATtiny817 board.

1.12. Boards
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

AVR-IoT WG Development Board

Contents

- AVR-IoT WG Development Board
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA4808</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>48KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use `avr_iot_wg` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:avr_iot_wg]
platform = atmelmegaavr
board = avr_iot_wg
```

You can override default AVR-IoT WG Development Board settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `avr_iot_wg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:avr_iot_wg]
platform = atmelmegaavr
board = avr_iot_wg
```

; change microcontroller
board_build.mcu = atmega4808
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*Debugging* currently does not support AVR-IoT WG Development Board board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Nano Every

Contents

- Arduino Nano Every
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA4809</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>47.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `nano_every` ID for `board` option in “platformio.ini” *(Project Configuration File)*:
You can override default Arduino Nano Every settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nano_every.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:nano_every]
platform = atmelmegaavr
board = nano_every

; change microcontroller
board_build.mcu = atmega4809

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

Debugging currently does not support Arduino Nano Every board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Arduino Uno WiFi Rev2**

Contents

- Arduino Uno WiFi Rev2
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.
## Configuration

Please use `uno_wifi_rev2` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:uno_wifi_rev2]
platform = atmelmegaavr
board = uno_wifi_rev2
```

You can override default Arduino Uno WiFi Rev2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `uno_wifi_rev2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:uno_wifi_rev2]
platform = atmelmegaavr
board = uno_wifi_rev2

; change microcontroller
board_build.mcu = atmega4809

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*Debugging* currently does not support Arduino Uno WiFi Rev2 board.

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Curiosity Nano ATmega4809

#### Contents

- *Curiosity Nano ATmega4809*
  - *Hardware*
  - *Configuration*
Hardware

Platform *Atmel megaAVR*: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA4809</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>48KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use `curiosity_nano_4809` ID for `board` option in “`platformio.ini`” (*Project Configuration File)*:

```ini
[env:curiosity_nano_4809]
platform = atmelmegaavr
board = curiosity_nano_4809
```

You can override default Curiosity Nano ATmega4809 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `curiosity_nano_4809.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:curiosity_nano_4809]
platform = atmelmegaavr
board = curiosity_nano_4809

; change microcontroller
board_build.mcu = atmega4809

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support Curiosity Nano ATmega4809 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Xplained Pro ATmega4809

Contents
- Xplained Pro ATmega4809
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel megaAVR**: 8-bit MCUs Built for Real-time Control with Core Independent Peripherals combining intelligent hardware peripherals along with the low-power capability of an AVR core, megaAVR microcontrollers (MCUs) broaden the effectiveness of your real-time control systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA4809</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>48KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use `xplained_pro_4809` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:xplained_pro_4809]
platform = atmelmegaavr
board = xplained_pro_4809
```

You can override default Xplained Pro ATmega4809 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xplained_pro_4809.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xplained_pro_4809]
platform = atmelmegaavr
board = xplained_pro_4809

; change microcontroller
board_build.mcu = atmega4809

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*Debugging* currently does not support Xplained Pro ATmega4809 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

1.12.5 Atmel SAM

Adafruit BLM Badge

Contents

- Adafruit BLM Badge
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21E18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_blm_badge ID for board option in “platformio.ini” (Project Configuration File):

```
[env:adafruit_blm_badge]
platform = atmel|sam
board = adafruit_blm_badge
```

You can override default Adafruit BLM Badge settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_blm_badge.json. For example, board_build.mcu, board_build.f_cpu, etc.
Adafruit BLM Badge supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_blm_badge]
platform = atmelsam
board = adafruit_blm_badge
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *platformio.ini* (Project Configuration File).

Adafruit BLM Badge does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Circuit Playground Express

Contents

- Adafruit Circuit Playground Express
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Atmel SAM*: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_circuitplayground_m0` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:adafruit_circuitplayground_m0]
platform = atmelsam
board = adafruit_circuitplayground_m0
```

You can override default Adafruit Circuit Playground Express settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_circuitplayground_m0.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:adafruit_circuitplayground_m0]
platform = atmelsam
board = adafruit_circuitplayground_m0
```

(continues on next page)
Uploading

Adafruit Circuit Playground Express supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_circuitplayground_m0]
platform = atmelsam
board = adafruit_circuitplayground_m0
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

Adafruit Circuit Playground Express does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Crickit M0

Contents

- Adafruit Crickit M0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_crickit_m0 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_crickit_m0]
platform = atmelsam
board = adafruit_crickit_m0
```

You can override default Adafruit Crickit M0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_crickit_m0.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:adafruit_crickit_m0]
platform = atmelsam
board = adafruit_crickit_m0
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

**Uploading**

Adafruit Crickit M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_crickit_m0]
platform = atmelsam
board = adafruit_crickit_m0
upload_protocol = sam-ba
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Adafruit Crickit M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Feather M0

Contents

- Adafruit Feather M0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_feather_m0 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:adafruit_feather_m0]
platform = atmelsam
board = adafruit_feather_m0
```

You can override default Adafruit Feather M0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_feather_m0.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:adafruit_feather_m0]
platform = atmelsam
board = adafruit_feather_m0
```

(continues on next page)
Uploading

Adafruit Feather M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_feather_m0]
platform = atmelsam
board = adafruit_feather_m0
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Feather M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Adafruit Feather M0 Express

Contents

- Adafruit Feather M0 Express
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_feather_m0_express ID for board option in "platformio.ini" (Project Configuration File):

```
[env:adafruit_feather_m0_express]
platform = atmel
board = adafruit_feather_m0_express
```

You can override default Adafruit Feather M0 Express settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_feather_m0_express.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:adafruit_feather_m0_express]
platform = atmel-sam
board = adafruit_feather_m0_express

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

Uploading

Adafruit Feather M0 Express supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:

[env:adafruit_feather_m0_express]
platform = atmel-sam
board = adafruit_feather_m0_express

upload_protocol = sam-ba

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit Feather M0 Express does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Feather M4 CAN

Contents

- Adafruit Feather M4 CAN
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAME51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>496KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_feather_m4_can` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:adafruit_feather_m4_can]
platform = atmelsam
board = adafruit_feather_m4_can
```

You can override default Adafruit Feather M4 CAN settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_feather_m4_can.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_feather_m4_can]
platform = atmelsam
board = adafruit_feather_m4_can
```

(continues on next page)
Uploading

Adafruit Feather M4 CAN supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_feather_m4_can]
platform = atmelsam
board = adafruit_feather_m4_can
upload_protocol = sam-ba
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Feather M4 CAN does not have on-board debug probe and is not ready for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Adafruit Feather M4 Express

Hardware

Platform *Atmel SAM*: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use *adafruit_feather_m4* ID for *board* option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:adafruit_feather_m4]
platform = atmelsam
board = adafruit_feather_m4
```

You can override default Adafruit Feather M4 Express settings per build environment using *board_*** option, where *** is a JSON object path from board manifest *adafruit_feather_m4.json*. For example, *board_build.mcu, board_build.f_cpu, etc*.

```ini
[env:adafruit_feather_m4]
platform = atmelsam
board = adafruit_feather_m4

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Feather M4 Express supports the next uploading protocols:
• atmel-ice
• jlink
• sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_feather_m4]
platform = atmelsam
board = adafruit_feather_m4
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

Adafruit Feather M4 Express does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

**Name** | **Description**
---|---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

### Adafruit Gemma M0

**Contents**

- *Adafruit Gemma M0*
  - *Hardware*
  - *Configuration*
  - *Uploading*
Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21E18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_gemma_m0 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_gemma_m0]
platform = atmelsam
board = adafruit_gemma_m0
```

You can override default Adafruit Gemma M0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_gemma_m0.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:adafruit_gemma_m0]
platform = atmelsam
board = adafruit_gemma_m0

; change microcontroller
board_build.mcu = samd21e18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Adafruit Gemma M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Gemma M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Grand Central M4

Contents

- Adafruit Grand Central M4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *Atmel SAM*: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51P20A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use *adafruit_grandcentral_m4* ID for *board* option in "platformio.ini” (*Project Configuration File*):

```
[env:adafruit_grandcentral_m4]
platform = atmelsam
board = adafruit_grandcentral_m4
```

You can override default Adafruit Grand Central M4 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *adafruit_grandcentral_m4.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:adafruit_grandcentral_m4]
platform = atmelsam
board = adafruit_grandcentral_m4

; change microcontroller
board_build.mcu = samd51p20a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Grand Central M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:adafruit_grandcentral_m4]
platform = atmelsam
board = adafruit_grandcentral_m4

upload_protocol = sam-ba
```
Debugging

"1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in "platformio.ini" (Project Configuration File).

Adafruit Grand Central M4 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Hallowing M0

Contents

- Adafruit Hallowing M0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.
## Configuration

Please use `adafruit_hallowing` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:adafruit_hallowing]
platform = atmelsam
board = adafruit_hallowing
```

You can override default Adafruit Hallowing M0 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_hallowing.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_hallowing]
platform = atmelsam
board = adafruit_hallowing

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

## Uploading

Adafruit Hallowing M0 supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_hallowing]
platform = atmelsam
board = adafruit_hallowing

upload_protocol = sam-ba
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (**Project Configuration File**).

Adafruit Hallowing M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Adafruit Hallowing M4**

**Contents**

- Adafruit Hallowing M4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>496KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>
Configuration

Please use `adafruit_hallowing_m4` ID for `board` option in "platformio.ini" (Project Configuration File):

```python
[env:adafruit_hallowing_m4]
platform = atmelsam
board = adafruit_hallowing_m4
```

You can override default Adafruit Hallowing M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_hallowing_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:adafruit_hallowing_m4]
platform = atmelsam
board = adafruit_hallowing_m4

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Hallowing M4 supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```python
[env:adafruit_hallowing_m4]
platform = atmelsam
board = adafruit_hallowing_m4

upload_protocol = sam-ba
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in "platformio.ini" (Project Configuration File).

Adafruit Hallowing M4 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Adafruit ItsyBitsy M0

**Contents**

- *Adafruit ItsyBitsy M0*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Atmel SAM*: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use *adafruit_itsybitsy_m0* ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:adafruit_itsybitsy_m0]
platform = atmelsam
board = adafruit_itsybitsy_m0
```

You can override default Adafruit ItsyBitsy M0 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_itsybitsy_m0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

Adafruit ItsyBitsy M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:

```
[env:adafruit_itsybitsy_m0]
platform = atmelsam
board = adafruit_itsybitsy_m0
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in *platformio.ini* (Project Configuration File).

Adafruit ItsyBitsy M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit ItsyBitsy M4

Contents

- Adafruit ItsyBitsy M4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51G19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_itsybitsy_m4 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:adafruit_itsybitsy_m4]
platform = atmelsam
board = adafruit_itsybitsy_m4
```

You can override default Adafruit ItsyBitsy M4 settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_itsybitsy_m4.json. For example, board_build.mcu, board_build.ffd_cpu, etc.

```
[env:adafruit_itsybitsy_m4]
platform = atmelsam
board = adafruit_itsybitsy_m4
```

(continues on next page)
; change microcontroller
board_build.mcu = samd51g19a

; change MCU frequency
board_build.f_cpu = 120000000L

---

**Uploading**

Adafruit ItsyBitsy M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_itsybitsy_m4]
platform = atmelsam
board = adafruit_itsybitsy_m4
upload_protocol = sam-ba
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

This chapter is about the debugging process and includes the following sections:

### Warning:
You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit ItsyBitsy M4 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Adafruit MONSTER M4SK

Contents

- Adafruit MONSTER M4SK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51G19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>496KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_monster_m4sk ID for board option in "platformio.ini" (Project Configuration File):

```
[env:adafruit_monster_m4sk]
platform = atmelsam
board = adafruit_monster_m4sk
```

You can override default Adafruit MONSTER M4SK settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_monster_m4sk.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

Adafruit MONSTER M4SK supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:

```
[env:adafruit_monster_m4sk]
platform = atmelsam
board = adafruit_monster_m4sk
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit MONSTER M4SK does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Atmel-ICE</em></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><em>J-LINK</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Matrix Portal M4

Contents

- Adafruit Matrix Portal M4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>496KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_matrix_portal_m4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:adafruit_matrix_portal_m4]
platform = atmel
board = adafruit_matrix_portal_m4
```

You can override default Adafruit Matrix Portal M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_matrix_portal_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_matrix_portal_m4]
platform = atmel
board = adafruit_matrix_portal_m4
```

(continues on next page)
;;; change microcontroller
board_build.mcu = samd51j19a

;;; change MCU frequency
board_build.f_cpu = 120000000L

---

### Uploading

Adafruit Matrix Portal M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is **sam-ba**

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_matrix_portal_m4]
platform = atmelsam
board = adafruit_matrix_portal_m4
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *platformio.ini* *(Project Configuration File)*.

Adafruit Matrix Portal M4 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Adafruit Metro M0 Expresss

Contents

- Adafruit Metro M0 Expresss
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_metro_m0` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:adafruit_metro_m0]
platform = atmelsam
board = adafruit_metro_m0
```

You can override default Adafruit Metro M0 Expresss settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_metro_m0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_metro_m0]
platform = atmelsam
board = adafruit_metro_m0

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Adafruit Metro M0 Expresss supports the next uploading protocols:
• atmel-ice
  • blackmagic
  • jlink
  • sam-ba

Default protocol is **sam-ba**

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruitMetro_m0]
platform = atmel-sam
board = adafruitMetro_m0
upload_protocol = sam-ba
```

**Debugging**

*Debugging - “1-click” solution for debugging with a zero configuration.*

**Warning**: You will need to install debugger tool drivers depending on your system. Please click on compatible debugger tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” **(Project Configuration File)**.

Adafruit Metro M0 Expresss does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Adafruit Metro M4**

**Contents**

• **Adafruit Metro M4**
  – **Hardware**
Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_metro_m4` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:adafruitMetro_m4]
platform = atmelsam
board = adafruitMetro_m4
```

You can override default Adafruit Metro M4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruitMetro_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruitMetro_m4]
platform = atmelsam
board = adafruitMetro_m4

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Metro M4 supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:
[env:adafruit_metro_m4]
platform = atmel-sam
board = adafruit_metro_m4
upload_protocol = sam-ba

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit Metro M4 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Metro M4 AirLift Lite

Contents

- Adafruit Metro M4 AirLift Lite
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_metro_m4_airliftlite` ID for `board` option in “platformio.ini” (Project Configuration File):

```python
[env:adafruit_metro_m4_airliftlite]
platform = atmelsam
board = adafruit_metro_m4_airliftlite
```

You can override default Adafruit Metro M4 AirLift Lite settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_metro_m4_airliftlite.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:adafruit_metro_m4_airliftlite]
platform = atmelsam
board = adafruit_metro_m4_airliftlite

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Metro M4 AirLift Lite supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```python
[env:adafruit_metro_m4_airliftlite]
platform = atmelsam
board = adafruit_metro_m4_airliftlite

upload_protocol = sam-ba
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit Metro M4 AirLift Lite does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit PyGamer Advance M4

Contents

- Adafruit PyGamer Advance M4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J20A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `adafruit_pygamer_advance_m4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:adafruit_pygamer_advance_m4]
platform = atmelsam
board = adafruit_pygamer_advance_m4
```

You can override default Adafruit PyGamer Advance M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_pygamer_advance_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_pygamer_advance_m4]
platform = atmelsam
board = adafruit_pygamer_advance_m4

; change microcontroller
board_build.mcu = samd51j20a

; change MCU frequency
board_build.f_cpu = 120000000L
```

**Uploading**

Adafruit PyGamer Advance M4 supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_pygamer_advance_m4]
platform = atmelsam
board = adafruit_pygamer_advance_m4

upload_protocol = sam-ba
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit PyGamer Advance M4 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit PyGamer M4 Express

Contents

- Adafruit PyGamer M4 Express
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>
Configuration

Please use `adafruit_pygamer_m4` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:adafruit_pygamer_m4]
platform = atmelsam
board = adafruit_pygamer_m4
```

You can override default Adafruit PyGamer M4 Express settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_pygamer_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_pygamer_m4]
platform = atmelsam
board = adafruit_pygamer_m4

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit PyGamer M4 Express supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_pygamer_m4]
platform = atmelsam
board = adafruit_pygamer_m4

upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit PyGamer M4 Express does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Adafruit PyPortal M4

#### Contents

- *Adafruit PyPortal M4*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Atmel SAM*: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J20A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

### Configuration

Please use `adafruit_pyportal_m4` ID for *board* option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_pyportal_m4]
platform = atmelsam
board = adafruit_pyportal_m4
```

You can override default Adafruit PyPortal M4 settings per build environment using *board_*** option, where *** is a JSON object path from board manifest `adafruit_pyportal_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:adafruit_pyportal_m4]
platform = atmelsam
board = adafruit_pyportal_m4

; change microcontroller
board_build.mcu = samd51j20a

; change MCU frequency
board_build.f_cpu = 120000000L

### Uploading

Adafruit PyPortal M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

[env:adafruit_pyportal_m4]
platform = atmelsam
board = adafruit_pyportal_m4

upload_protocol = sam-ba

### Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit PyPortal M4 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit PyPortal M4 Titano

Contents

- Adafruit PyPortal M4 Titano
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J20A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_pyportal_m4_titano` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:adafruit_pyportal_m4_titano]
platform = atmelsam
board = adafruit_pyportal_m4_titano
```

You can override default Adafruit PyPortal M4 Titano settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_pyportal_m4_titano.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_pyportal_m4_titano]
platform = atmelsam
board = adafruit_pyportal_m4_titano
```

(continues on next page)
Uploading

Adafruit PyPortal M4 Titano supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_pyportal_m4_titano]
platform = atmelsam
board = adafruit_pyportal_m4_titano
upload_protocol = sam-ba
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit PyPortal M4 Titano does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Adafruit QT Py M0

Contents

- Adafruit QT Py M0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21E18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use adafruit_qt_py_m0 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:adafruit_qt_py_m0]
platform = atmelsam
board = adafruit_qt_py_m0
```

You can override default Adafruit QT Py M0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest adafruit_qt_py_m0.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:adafruit_qt_py_m0]
platform = atmelsam
board = adafruit_qt_py_m0

; change microcontroller
board_build.mcu = samd21e18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Adafruit QT Py M0 supports the next uploading protocols:
• atmel-ice
• blackmagic
• jlink
• sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```yaml
[env:adafruit_qt_py_m0]
platform = atmel-sam
board = adafruit_qt_py_m0
upload_protocol = sam-ba
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit QT Py M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Adafruit Trellis M4

**Contents**

- **Adafruit Trellis M4**
  - Hardware
Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_trellis_m4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:adafruit_trellis_m4]
platform = atmelsam
board = adafruit_trellis_m4
```

You can override default Adafruit Trellis M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_trellis_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_trellis_m4]
platform = atmelsam
board = adafruit_trellis_m4

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Trellis M4 supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:
[env:adafruit_trellis_m4]
platform = atmelsam
board = adafruit_trellis_m4
upload_protocol = sam-ba

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Trellis M4 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Trinket M0

Contents

- *Adafruit Trinket M0*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21E18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_trinket_m0` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_trinket_m0]
platform = atmelsam
board = adafruit_trinket_m0
```

You can override default Adafruit Trinket M0 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_trinket_m0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_trinket_m0]
platform = atmelsam
board = adafruit_trinket_m0

; change microcontroller
board_build.mcu = samd21e18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Adafruit Trinket M0 supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_trinket_m0]
platform = atmelsam
board = adafruit_trinket_m0

upload_protocol = sam-ba
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Adafruit Trinket M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Adafruit pIRkey

Contents

- Adafruit pIRkey
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.
### Configuration

Please use `adafruit_pirkey` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:adafruit_pirkey]
platform = atmelsam
board = adafruit_pirkey
```

You can override default Adafruit pIRkey settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_pirkey.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.:

```
[env:adafruit_pirkey]
platform = atmelsam
board = adafruit_pirkey

; change microcontroller
board_build.mcu = samd21e18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Adafruit pIRkey supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_pirkey]
platform = atmelsam
board = adafruit_pirkey

upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit pIRkey does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Adafruit pyBadge AirLift M4**

**Contents**

- Adafruit pyBadge AirLift M4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J20A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1008KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>
Configuration

Please use `adafruit_pybadge_airlift_m4` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_pybadge_airlift_m4]
platform = atmelsam
board = adafruit_pybadge_airlift_m4
```

You can override default Adafruit pyBadge AirLift M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_pybadge_airlift_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_pybadge_airlift_m4]
platform = atmelsam
board = adafruit_pybadge_airlift_m4

; change microcontroller
board_build.mcu = samd51j20a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit pyBadge AirLift M4 supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba` You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_pybadge_airlift_m4]
platform = atmelsam
board = adafruit_pybadge_airlift_m4

upload_protocol = sam-ba
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).
Adafruit pyBadge AirLift M4 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Adafruit pyBadge M4 Express**

**Contents**

- Adafruit pyBadge M4 Express
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use **adafruit_pybadge_m4** ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_pybadge_m4]
platform = atmelsam
board = adafruit_pybadge_m4
```
You can override default Adafruit pyBadge M4 Express settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_pybadge_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_pybadge_m4]
platform = atmelsam
board = adafruit_pybadge_m4

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

## Uploading

Adafruit pyBadge M4 Express supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_pybadge_m4]
platform = atmelsam
board = adafruit_pybadge_m4

upload_protocol = sam-ba
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “`platformio.ini` (Project Configuration File)

Adafruit pyBadge M4 Express does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-Link</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Due (Programming Port)

Contents

- Arduino Due (Programming Port)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT91SAM3X8E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use due ID for board option in "platformio.ini" (Project Configuration File):

```
[env:due]
platform = atmelssam
board = due
```

You can override default Arduino Due (Programming Port) settings per build environment using board_*** option, where *** is a JSON object path from board manifest due.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:due]
platform = atmelssam
board = due
```

(continues on next page)
Arduino Due (Programming Port) supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba
- stlink

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:due]
platform = atmelsam
board = due
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Due (Programming Port) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Arduino Due (USB Native Port)

Contents

- Arduino Due (USB Native Port)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT91SAM3X8E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use dueUSB ID for board option in “platformio.ini” (Project Configuration File):

```
[env:dueUSB]
platform = atmel
board = dueUSB
```

You can override default Arduino Due (USB Native Port) settings per build environment using board_*** option, where *** is a JSON object path from board manifest dueUSB.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

Arduino Due (USB Native Port) supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba
- stlink

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:dueUSB]
platform = atmelsam
board = dueUSB

upload_protocol = sam-ba
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Due (USB Native Port) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

Arduino M0

Contents

- Arduino M0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mzeroUSB ID for board option in “platformio.ini” (Project Configuration File): 

```
[env:mzeroUSB]
platform = atmelsam
board = mzeroUSB
```

You can override default Arduino M0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest mzeroUSB.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mzeroUSB]
platform = atmelsam
board = mzeroUSB
```

(continues on next page)
Uploading

Arduino M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- stk500v2

Default protocol is stk500v2

You can change upload protocol using upload_protocol option:

```ini
[env:mzeroUSB]
platform = atmel-sam
board = mzeroUSB
upload_protocol = stk500v2
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in "platformio.ini" (Project Configuration File).

Arduino M0 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino M0 Pro (Native USB Port)

Contents

- Arduino M0 Pro (Native USB Port)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mzero pro USB ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:mzero proUSB]
platform = atmelsam
board = mzero proUSB
```

You can override default Arduino M0 Pro (Native USB Port) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest mzero proUSB.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mzero proUSB]
platform = atmelsam
board = mzero proUSB
```

(continues on next page)
Uploading

Arduino M0 Pro (Native USB Port) supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- stk500v2

Default protocol is stk500v2

You can change upload protocol using `upload_protocol` option:

```ini
[env:mzeroproUSB]
platform = atmel-sam
board = mzeroproUSB
upload_protocol = stk500v2
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Arduino M0 Pro (Native USB Port) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino M0 Pro (Programming/Debug Port)

Contents

- Arduino M0 Pro (Programming/Debug Port)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mzeropro ID for board option in “platformio.ini” (Project Configuration File):

```
[env:mzeropro]
platform = atmelsam
board = mzeropro
```

You can override default Arduino M0 Pro (Programming/Debug Port) settings per build environment using board_*** option, where *** is a JSON object path from board manifest mzeropro.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mzeropro]
platform = atmelsam
board = mzeropro
```

(continues on next page)
Uploading

Arduino M0 Pro (Programming/Debug Port) supports the next uploading protocols:

- atmel-ice
- blackmagic
- cmsis-dap
- jlink

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```
[env:mzeropro]
platform = atmelsam
board = mzeropro
upload_protocol = cmsis-dap
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (*Project Configuration File)*.

Arduino M0 Pro (Programming/Debug Port) has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino MKR FOX 1200

Contents

- Arduino MKR FOX 1200
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mkrfoxl200 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:mkrfox1200]
platform = atmelsam
board = mkrfoxl200
```

You can override default Arduino MKR FOX 1200 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `mkrfox1200.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mkrfox1200]
platform = atmelsam
board = mkrfoxl200
```

(continues on next page)
Uploading

Arduino MKR FOX 1200 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:

```
[env:mkrfox1200]
platform = atmelsam
board = mkrfox1200
upload_protocol = sam-ba
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Arduino MKR FOX 1200 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino MKR GSM 1400

Contents

- Arduino MKR GSM 1400
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mkrgsm1400 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:mkrgsm1400]
platform = atmel|sam
board = mkrgsm1400
```

You can override default Arduino MKR GSM 1400 settings per build environment using board_*** option, where *** is a JSON object path from board manifest mkrgsm1400.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:mkrgsm1400]
platform = atmel|sam
board = mkrgsm1400
```

(continues on next page)
}; change microcontroller
board_build.mcu = samd21g18a

}; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

Arduino MKR GSM 1400 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:mkrgsm1400]
platform = atmelsam
board = mkrgsm1400
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino MKR GSM 1400 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino MKR NB 1500

Contents

- Arduino MKR NB 1500
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mkrnb1500 ID for board option in “platformio.ini” (Project Configuration File):

```python
[env:mkrnb1500]
platform = atmel
board = mkrnb1500
```

You can override default Arduino MKR NB 1500 settings per build environment using board_*** option, where *** is a JSON object path from board manifest mkrnb1500.json. For example, board_build.mcu, board_build.f_cpu, etc.

```python
[env:mkrnb1500]
platform = atmel
board = mkrnb1500
```

(continues on next page)
PlatformIO Documentation, Release 5.0.5a1

(continued from previous page)

```plaintext
}; change microcontroller
board_build.mcu = samd21g18a

}; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Arduino MKR NB 1500 supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:mkrnb1500]
platform = atmelsam
board = mkrnb1500

upload_protocol = sam-ba
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino MKR NB 1500 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Atmel-ICE</em></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><em>Black Magic Probe</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>J-LINK</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino MKR WAN 1300

Contents

- Arduino MKR WAN 1300
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mkrwan1300 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:mkrwan1300]
platform = atmel
board = mkrwan1300
```

You can override default Arduino MKR WAN 1300 settings per build environment using board_*** option, where *** is a JSON object path from board manifest mkrwan1300.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mkrwan1300]
platform = atmel
board = mkrwan1300
```

(continues on next page)
Uploading

Arduino MKR WAN 1300 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:mkrwan1300]
platform = atmelsam
board = mkrwan1300
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Arduino MKR WAN 1300 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Atmel-ICE</em></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><em>Black Magic Probe</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>J-LINK</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino MKR WAN 1310

Contents

- Arduino MKR WAN 1310
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mkrwan1310 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:mkrwan1310]
platform = atmelsam
board = mkrwan1310
```

You can override default Arduino MKR WAN 1310 settings per build environment using board_*** option, where *** is a JSON object path from board manifest mkrwan1310.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mkrwan1310]
platform = atmelsam
board = mkrwan1310
```

(continues on next page)
Uploading

Arduino MKR WAN 1310 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:mkrwan1310]
platform = atmel-sam
board = mkrwan1310
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Arduino MKR WAN 1310 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino MKR WiFi 1010

Contents

- Arduino MKR WiFi 1010
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mkrwifi1010 ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:mkrwifi1010]
platform = atmelsam
board = mkrwifi1010
```

You can override default Arduino MKR WiFi 1010 settings per build environment using board_*** option, where *** is a JSON object path from board manifest mkrwifi1010.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:mkrwifi1010]
platform = atmelsam
board = mkrwifi1010
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

## Uploading

Arduino MKR WiFi 1010 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:mkrwifi1010]
platform = atmel-sam
board = mkrwifi1010
upload_protocol = sam-ba
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.

Arduino MKR WiFi 1010 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Atmel-ICE</em></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><em>Black Magic Probe</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>J-LINK</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Arduino MKR1000**

**Contents**

- Arduino MKR1000
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use mkr1000USB ID for *board* option in “platformio.ini” *(Project Configuration File):*

```
[env:mkr1000USB]
platform = atmelsam
board = mkr1000USB
```

You can override default Arduino MKR1000 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest mkr1000USB.json. For example, `board_build.mcu`, `board_build. f_cpu`, etc.

```
[env:mkr1000USB]
platform = atmelsam
board = mkr1000USB
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

Arduino MKR1000 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:mkr1000USB]
platform = atmelsam
board = mkr1000USB
upload_protocol = sam-ba
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino MKR1000 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino MKRZERO

Contents

- Arduino MKRZERO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use mkrzero ID for board option in “platformio.ini” (Project Configuration File):

```
[env:mkrzero]
platform = atmel_sam
board = mkrzero
```

You can override default Arduino MKRZERO settings per build environment using board_*** option, where *** is a JSON object path from board manifest mkrzero.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mkrzero]
platform = atmel_sam
board = mkrzero
```

(continues on next page)
Uploaded Arduino MKRZERO supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:mkrzero]
platform = atmelsam
board = mkrzero
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

*Warning:* You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Arduino MKRZERO does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Atmel-ICE</em></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><em>Black Magic Probe</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>J-LINK</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Arduino Tian

Contents

- Arduino Tian
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel l SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use tian ID for board option in “platformio.ini” (Project Configuration File):

```
[env:tian]
platform = atmel
board = tian
```

You can override default Arduino Tian settings per build environment using board_*** option, where *** is a JSON object path from board manifest tian.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:tian]
platform = atmel
board = tian

; change microcontroller
```

(continues on next page)
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

Uploading

Arduino Tian supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- stk500v2

Default protocol is stk500v2

You can change upload protocol using upload_protocol option:

```
[env:tian]
platform = atmel-sam
board = tian
upload_protocol = stk500v2
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Arduino Tian does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Arduino Zero (Programming/Debug Port)

Contents

- Arduino Zero (Programming/Debug Port)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use zero ID for board option in "platformio.ini" (Project Configuration File):

```
[env:zero]
platform = atmelsam
board = zero
```

You can override default Arduino Zero (Programming/Debug Port) settings per build environment using board_*** option, where *** is a JSON object path from board manifest zero.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:zero]
platform = atmelsam
board = zero

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Arduino Zero (Programming/Debug Port) supports the next uploading protocols:
• atmel-ice
• blackmagic
• cmsis-dap
• jlink

Default protocol is cmsis-dap

You can change upload protocol using `upload_protocol` option:

```ini
[env:zero]
platform = atmelsam
board = zero
upload_protocol = cmsis-dap
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Zero (Programming/Debug Port) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

**Arduino Zero (USB Native Port)**
Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use zeroUSB ID for board option in “platformio.ini” (Project Configuration File):

```
[env:zeroUSB]
platform = atmel-sam
board = zeroUSB
```

You can override default Arduino Zero (USB Native Port) settings per build environment using board_*** option, where *** is a JSON object path from board manifest zeroUSB.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:zeroUSB]
platform = atmel-sam
board = zeroUSB

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Arduino Zero (USB Native Port) supports the next uploading protocols:

- atmel-ice
- blackmagic
• jlink
• sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:zeroUSB]
platform = atmelsam
board = zeroUSB

upload_protocol = sam-ba
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Zero (USB Native Port) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Atmel ATSAMR21-XPRO**

**Contents**

• *Atmel ATSAMR21-XPRO*
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMR21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `samr21_xpro` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:samr21_xpro]
platform = atmelsam
board = samr21_xpro
```

You can override default Atmel ATSAMR21-XPRO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `samr21_xpro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:samr21_xpro]
platform = atmelsam
board = samr21_xpro

; change microcontroller
board_build.mcu = samr21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Atmel ATSAMR21-XPRO supports the next uploading protocols:

- atmel-ice
- blackmagic
- cmsis-dap
- jlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (**Project Configuration File**).

Atmel ATSAMR21-XPRO has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Atmel ATSAMW25-XPRO

Contents

- **Atmel ATSAMW25-XPRO**
  - **Hardware**
  - **Configuration**
  - **Uploading**
  - **Debugging**
  - **Frameworks**
Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `samd21g18a` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:samd21g18a]
platform = atmelsam
board = samd21g18a
```

You can override default Atmel ATSAMW25-XPRO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `samd21g18a.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:samd21g18a]
platform = atmelsam
board = samd21g18a

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Atmel ATSAMW25-XPRO supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `cmsis-dap`
- `jlink`

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:samd21g18a]
platform = atmelsam
board = samd21g18a

upload_protocol = cmsis-dap
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Atmel ATSAMW25-XPRO has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**Atmel SAMC21-XPRO**

**Contents**

- *Atmel SAMC21-XPRO*
  - Hardware
  - Configuration
  - Uploading
  - Debugging

**Hardware**

Platform *Atmel SAM*: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMC21J18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `samc21_xpro` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:samc21_xpro]
platform = atmelsam
board = samc21_xpro
```

You can override default Atmel SAMC21-XPRO settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `samc21_xpro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:samc21_xpro]
platform = atmelsam
board = samc21_xpro

; change microcontroller
board_build.mcu = samc21j18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

Atmel SAMC21-XPRO supports the next uploading protocols:

- atmel-ice
- blackmagic
- cmsis-dap
- jlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:samc21_xpro]
platform = atmelsam
board = samc21_xpro

upload_protocol = cmsis-dap
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Atmel SAMC21-XPRO has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Atmel SAMD21-XPRO

Contents

- Atmel SAMD21-XPRO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21J18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use samd21_xpro ID for board option in “platformio.ini” (Project Configuration File):

```
[env:samd21_xpro]
platform = atmelsam
board = samd21_xpro
```
You can override default Atmel SAMD21-XPRO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `samd21_xpro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:samd21_xpro]
platform = atmelsam
board = samd21_xpro

; change microcontroller
board_build.mcu = samd21j18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Atmel SAMD21-XPRO supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `cmsis-dap`
- `jlink`

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:samd21_xpro]
platform = atmelsam
board = samd21_xpro

upload_protocol = cmsis-dap
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

Atmel SAMD21-XPRO has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Atmel-ICE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CMSIS-DAP</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>J-LINK</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Atmel SAML21-XPRO-B

Contents

- Atmel SAML21-XPRO-B
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAML21J18B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use saml21_xpro_b ID for board option in “platformio.ini” (Project Configuration File):

```
[env:saml21_xpro_b]
platform = atmelsam
board = saml21_xpro_b
```

You can override default Atmel SAML21-XPRO-B settings per build environment using board_*** option, where *** is a JSON object path from board manifest saml21_xpro_b.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:saml21_xpro_b]
platform = atmel-sam
board = saml21_xpro_b

; change microcontroller
board_build.mcu = saml21j18b

; change MCU frequency
board_build.f_cpu = 48000000L

## Uploading

Atmel SAML21-XPRO-B supports the next uploading protocols:

- atmel-ice
- blackmagic
- cmsis-dap
- jlink

Default protocol is cmsis-dap

You can change upload protocol using `upload_protocol` option:

[env:saml21_xpro_b]
platform = atmel-sam
board = saml21_xpro_b

upload_protocol = cmsis-dap

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Atmel SAML21-XPRO-B has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Briki ABC (MBC-WB) - Samd21

Contents

- Briki ABC (MBC-WB) - Samd21
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>meteca</td>
</tr>
</tbody>
</table>

Configuration

Please use `briki_abc_samd21` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:briki_abc_samd21]
platform = atmel
board = briki_abc_samd21
```

You can override default Briki ABC (MBC-WB) - Samd21 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `briki_abc_samd21.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:briki_abc_samd21]
platform = atmel
board = briki_abc_samd21
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

Briki ABC (MBC-WB) - Samd21 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- mbctool
- sam-ba
- stk500v2

Default protocol is mbctool

You can change upload protocol using `upload_protocol` option:

```
[env:briki_abc_samd21]
platform = atmelsam
board = briki_abc_samd21
upload_protocol = mbctool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in `platformio.ini` (Project Configuration File).

Briki ABC (MBC-WB) - Samd21 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Briki MBC-WB - Samd21

Contents

- Briki MBC-WB - Samd21
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>meteca</td>
</tr>
</tbody>
</table>

Configuration

Please use briki_mbcwb_samd21 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:briki_mbcwb_samd21]
platform = atmelsam
board = briki_mbcwb_samd21
```

You can override default Briki MBC-WB - Samd21 settings per build environment using board_*** option, where *** is a JSON object path from board manifest briki_mbcwb_samd21.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:briki_mbcwb_samd21]
platform = atmelsam
board = briki_mbcwb_samd21
```

(continues on next page)
Uploading

Briki MBC-WB - Samd21 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- mbctool
- sam-ba
- stk500v2

Default protocol is mbctool

You can change upload protocol using upload_protocol option:

```plaintext
[env:briki_mbcwb_samd21]
platform = atmelsam
board = briki_mbcwb_samd21
upload_protocol = mbctool
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in "platformio.ini" (Project Configuration File).

Briki MBC-WB - Samd21 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digistump DigiX

Contents

- Digistump DigiX
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT91SAM3X8E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digistump</td>
</tr>
</tbody>
</table>

Configuration

Please use digix ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:digix]
platform = atmel
board = digix
```

You can override default Digistump DigiX settings per build environment using board_*** option, where *** is a JSON object path from board manifest digix.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:digix]
platform = atmel
board = digix
```

(continues on next page)
Uploading

Digistump DigiX supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`
- `stlink`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:digix]
platform = atmelsam
board = digix
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Digistump DigiX does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

LowPowerLab CurrentRanger

Contents

- LowPowerLab CurrentRanger
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LowPowerLab</td>
</tr>
</tbody>
</table>

Configuration

Please use `current_ranger` ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:current_ranger]
platform = atmelsam
board = current_ranger
```

You can override default LowPowerLab CurrentRanger settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `current_ranger.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:current_ranger]
platform = atmelsam
board = current_ranger

; change microcontroller
board_build.mcu = samd21g18a
```

(continues on next page)
; change MCU frequency
board_build.f_cpu = 48000000L

Debugging

Debugging currently does not support LowPowerLab CurrentRanger board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

MKR Vidor 4000

Contents

- MKR Vidor 4000
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Atmel SAM*: Atmel SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `mkrvidor4000` ID for `board` option in “platformio.ini” *(Project Configuration File)*:
You can override default MKR Vidor 4000 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mkrvidor4000.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mkrvidor4000]
platform = atmelsam
board = mkrvidor4000

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

MKR Vidor 4000 supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:mkrvidor4000]
platform = atmelsam
board = mkrvidor4000

upload_protocol = sam-ba
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” *(Project Configuration File)*.

MKR Vidor 4000 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Minitronics v2.0**

**Contents**

- Minitronics v2.0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21J18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ReprapWorld</td>
</tr>
</tbody>
</table>

**Configuration**

Please use minitronics20 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:minitronics20]
platform = atmelSam
board = minitronics20
```
You can override default Minitronics v2.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `minitronics20.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```bash
[env:minitronics20]
platform = atmel.sam
board = minitronics20

; change microcontroller
board_build.mcu = samd21j18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Minitronics v2.0 supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```bash
[env:minitronics20]
platform = atmel.sam
board = minitronics20

upload_protocol = sam-ba
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Minitronics v2.0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Atmel-ICE</code></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><code>Black Magic Probe</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>J-LINK</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Moteino M0

Contents

- Moteino M0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LowPowerLab</td>
</tr>
</tbody>
</table>

Configuration

Please use moteino_zero ID for board option in “platformio.ini” (Project Configuration File):

```
[env:moteino_zero]
platform = atmel | sam
board = moteino_zero
```

You can override default Moteino M0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest moteino_zero.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:moteino_zero]
platform = atmel | sam
board = moteino_zero
```

(continues on next page)
Uploading

Moteino M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- cmsis-dap
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:moteino_zero]
platform = atmel-atmega32u4
board = moteino_zero
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Moteino M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

NANO 33 IoT

Contents

- NANO 33 IoT
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Arduino</td>
</tr>
</tbody>
</table>

Configuration

Please use nano_33_iot ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nano_33_iot]
platform = atmelsam
board = nano_33_iot
```

You can override default NANO 33 IoT settings per build environment using board_*** option, where *** is a JSON object path from board manifest nano_33_iot.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nano_33_iot]
platform = atmelsam
board = nano_33_iot
```

(continues on next page)
Uploading

NANO 33 IoT supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:nano_33_iot]
platform = atmelsam
board = nano_33_iot
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File).*

NANO 33 IoT does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
</tbody>
</table>

SODAQ Autonomo

Contents

- **SODAQ Autonomo**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21J18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use `sodaq_autonomo` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:sodaq_autonomo]
platform = atmelsam
board = sodaq_autonomo
```

You can override default SODAQ Autonomo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sodaq_autonomo.json`. For example, `board_build.mcu,board_build.f_cpu, etc.`
[env:sodaq_autonomo]
platform = atmel-sam
board = sodaq_autonomo

; change microcontroller
board_build.mcu = samd21j18a

; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

SODAQ Autonomo supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

[env:sodaq_autonomo]
platform = atmel-sam
board = sodaq_autonomo

upload_protocol = sam-ba

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *`platformio.ini` (Project Configuration File)*.

SODAQ Autonomo does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td><strong>Yes</strong></td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ ExpLoRer

Contents

- SODAQ ExpLoRer
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21J18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use `sodaq_explorer` ID for `board` option in “`platformio.ini`” (**Project Configuration File**):

```ini
[env:sodaq_explorer]
platform = atmelsam
board = sodaq_explorer
```

You can override default SODAQ ExpLoRer settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sodaq_explorer.json`. For example, `board_build.mcu,board_build.f_cpu, etc.`

```ini
[env:sodaq_explorer]
platform = atmelsam
board = sodaq_explorer
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21j18a

; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

SODAQ ExpLoRer supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:sodaq_explorer]
platform = atmel-sam
board = sodaq_explorer
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

SODAQ ExpLoRer does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ ONE

Contents

- **SODAQ ONE**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use `sodaq_one` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:sodaq_one]
platform = atmel
board = sodaq
```

You can override default SODAQ ONE settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sodaq_one.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sodaq_one]
platform = atmel
board = sodaq_one
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

SODAQ ONE supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:sodaq_one]
platform = atmelsam
board = sodaq_one
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

SODAQ ONE does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ SARA

Contents

- **SODAQ SARA**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21J18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use `sodaq_sara` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:sodaq_sara]
platform = atmelsam
board = sodaq_sara
```

You can override default SODAQ SARA settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sodaq_sara.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:sodaq_sara]
platform = atmelsam
board = sodaq_sara
```

(continues on next page)
Uploads

SODAQ SARA supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```ini
[env:sodaq_sara]
platform = atmelsam
board = sodaq_sara
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

SODAQ SARA does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SODAQ SFF

Contents

- SODAQ SFF
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel l SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SODAQ</td>
</tr>
</tbody>
</table>

Configuration

Please use sodaq_sff ID for board option in “platformio.ini” (Project Configuration File):

```plaintext
[env:sodaq_sff]
platform = atmelsam
board = sodaq_sff
```

You can override default SODAQ SFF settings per build environment using board_*** option, where *** is a JSON object path from board manifest sodaq_sff.json. For example, board_build.mcu, board_build.f_cpu, etc.

```plaintext
[env:sodaq_sff]
platform = atmelsam
board = sodaq_sff

; change microcontroller
```

(continues on next page)
board_build.mcu = samd21g18a
; change MCU frequency
board_build.f_cpu = 48000000L

Uploading

SODAQ SFF supports the next uploading protocols:
   • atmel-ice
   • blackmagic
   • jlink
   • sam-ba

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:

```
[env:sodaq_sff]
platform = atmelsam
board = sodaq_sff
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” *(Project Configuration File)*.

SODAQ SFF does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
SainSmart Due (Programming Port)

Contents

• SainSmart Due (Programming Port)
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT91SAM3X8E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SainSmart</td>
</tr>
</tbody>
</table>

Configuration

Please use sainSmartDue ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:sainSmartDue]
platform = atmelsam
board = sainSmartDue
```

You can override default SainSmart Due (Programming Port) settings per build environment using board_*** option, where *** is a JSON object path from board manifest sainSmartDue.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:sainSmartDue]
platform = atmelsam
board = sainSmartDue

; change microcontroller
board_build.mcu = at91sam3x8e

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

SainSmart Due (Programming Port) supports the next uploading protocols:
PlatformIO Documentation, Release 5.0.5a1

- atmel-ice
- blackmagic
- jlink
- sam-ba
- stlink

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:sainSmartDue]
platform = atmelsam
board = sainSmartDue
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SainSmart Due (Programming Port) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SainSmart Due (USB Native Port)

Contents
• SainSmart Due (USB Native Port)
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform *Atmel SAM*: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT91SAM3X8E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SainSmart</td>
</tr>
</tbody>
</table>

Configuration

Please use sainSmartDueUSB ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sainSmartDueUSB]
platform = atmelsam
board = sainSmartDueUSB
```

You can override default SainSmart Due (USB Native Port) settings per build environment using board_*** option, where *** is a JSON object path from board manifest sainSmartDueUSB.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sainSmartDueUSB]
platform = atmelsam
board = sainSmartDueUSB

; change microcontroller
board_build.mcu = at91sam3x8e

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

SainSmart Due (USB Native Port) supports the next uploading protocols:

• atmel-ice
• blackmagic
• jlink
• sam-ba
• stlink

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:

```
[env:sainSmartDueUSB]
platform = atmelSam
board = sainSmartDueUSB
upload_protocol = sam-ba
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

SainSmart Due (USB Native Port) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Seesduino Femto M0

**Contents**

• *Seesduino Femto M0*
  - Hardware
  - Configuration
Hardware

Platform **Atmel SAM**: Atmel SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Seeed</td>
</tr>
</tbody>
</table>

Configuration

Please use `seeed_femto` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:seeed_femto]
platform = atmelsam
board = seeed_femto
```

You can override default Seeeduino Femto M0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `seeed_femto.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:seeed_femto]
platform = atmelsam
board = seeed_femto

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Seeeduino Femto M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:
[env:seeed_femto]
platform = atmelsam
board = seeed_femto
upload_protocol = sam-ba

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Seeeduino Femto M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Seeeduino LoRaWAN

Contents

- *Seeeduino LoRaWAN*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Seeed</td>
</tr>
</tbody>
</table>

Configuration

Please use `seeeduino_lorawan` ID for **board** option in “platformio.ini” (Project Configuration File):

```
[env:seeeduino_lorawan]
platform = atmelsam
board = seeeduino_lorawan
```

You can override default Seeeduino LoRaWAN settings per build environment using **board_*** option, where *** is a JSON object path from board manifest `seeeduino_lorawan.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:seeeduino_lorawan]
platform = atmelsam
board = seeeduino_lorawan

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Seeeduino LoRaWAN supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using **upload_protocol** option:

```
[env:seeeduino_lorawan]
platform = atmelsam
board = seeeduino_lorawan

upload_protocol = sam-ba
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Seeeduino LoRaWAN does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

**Name** | **Description**
---|---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

Seeeduino Wio Lite MG126

Contents

- Seeeduino Wio Lite MG126
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Seeed</td>
</tr>
</tbody>
</table>

### Configuration

Please use `seeed_wio_lite_mg126` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:seeed_wio_lite_mg126]
platform = atmel sam
board = seeed_wio_lite_mg126
```

You can override default Seeeduino Wio Lite MG126 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `seeed_wio_lite_mg126.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:seeed_wio_lite_mg126]
platform = atmel sam
board = seeed_wio_lite_mg126

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Seeeduino Wio Lite MG126 supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:seeed_wio_lite_mg126]
platform = atmel sam
board = seeed_wio_lite_mg126

upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in "platformio.ini" (Project Configuration File).

Seeeduino Wio Lite MG126 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Seeeduino Wio Terminal

#### Contents

- **Seeeduino Wio Terminal**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51P19A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>496KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Seeed</td>
</tr>
</tbody>
</table>
Configuration

Please use `seeed_wio_terminal` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:seeed_wio_terminal]
platform = atmelsam
board = seeed_wio_terminal
```

You can override default Seeeduino Wio Terminal settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `seeed_wio_terminal.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:seeed_wio_terminal]
platform = atmelsam
board = seeed_wio_terminal

; change microcontroller
board_build.mcu = samd51p19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Seeeduino Wio Terminal supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:seeed_wio_terminal]
platform = atmelsam
board = seeed_wio_terminal

upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).
Seeeduino Wio Terminal does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Seeeduino XIAO

#### Contents

- Seeeduino XIAO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Seeed</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `seeed_xiao` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:seeed_xiao]
platform = atmelSAM
board = seeed_xiao
```
You can override default Seeeduino XIAO settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `seeed_xiao.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:seeed_xiao]
platform = atmelsam
board = seeed_xiao

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Seeeduino XIAO supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:seeed_xiao]
platform = atmelsam
board = seeed_xiao

upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Seeeduino XIAO does not have on-board debug probe and is **NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
</tbody>
</table>

Seeeduino Zero

Contents

- Seeeduino Zero
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Seeed</td>
</tr>
</tbody>
</table>

Configuration

Please use `seeed_zero` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:seeed_zero]
platform = atmelsam
board = seeed_zero
```

You can override default Seeeduino Zero settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `seeed_zero.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

Seeeduino Zero supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:seeed_zero]
platform = atmelsam
board = seeed_zero
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Seeeduino Zero does not have on-board debug probe and *IS NOT READY* for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun 9DoF Razor IMU M0

Contents

- SparkFun 9DoF Razor IMU M0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_samd21_9dof ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_samd21_9dof]
platform = atmelsam
board = sparkfun_samd21_9dof
```

You can override default SparkFun 9DoF Razor IMU M0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_samd21_9dof.json. For example, board_build.
mcu, board_build.f_cpu, etc.

```
[env:sparkfun_samd21_9dof]
platform = atmelsam
board = sparkfun_samd21_9dof
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L

Uploading

SparkFun 9DoF Razor IMU M0 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using upload_protocol option:

[env:sparkfun_samd21_9dof]
platform = atmel-sam
board = sparkfun_samd21_9dof
upload_protocol = sam-ba

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

SparkFun 9DoF Razor IMU M0 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun Qwiic Micro

Contents

- SparkFun Qwiic Micro
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21E18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_qwiic_micro_samd21e ID for board option in “platformio.ini” (Project Configuration File):

```python
[env:sparkfun_qwiic_micro_samd21e]
platform = atmelsam
board = sparkfun_qwiic_micro_samd21e
```

You can override default SparkFun Qwiic Micro settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_qwiic_micro_samd21e.json. For example, board_build.mcuf, board_build.f_cpu, etc.

```python
[env:sparkfun_qwiic_micro_samd21e]
platform = atmelsam
board = sparkfun_qwiic_micro_samd21e
```

(continues on next page)
Uploading

SparkFun Qwiic Micro supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:sparkfun_qwiic_micro_samd21e]
platform = atmelsam
board = sparkfun_qwiic_micro_samd21e
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun Qwiic Micro does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun RedBoard Turbo

Contents

- SparkFun RedBoard Turbo
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_redboard_turbo ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_redboard_turbo]
platform = atmelsam
board = sparkfun_redboard_turbo
```

You can override default SparkFun RedBoard Turbo settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_redboard_turbo.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sparkfun_redboard_turbo]
platform = atmelsam
board = sparkfun_redboard_turbo
```

(continues on next page)
```plaintext
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

## Uploading

SparkFun RedBoard Turbo supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:sparkfun_redboard_turbo]
platform = atmelsam
board = sparkfun_redboard_turbo
upload_protocol = sam-ba
```

## Debugging

*Debugging*- “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun RedBoard Turbo does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun SAMD21 Dev Breakout

Contents

- SparkFun SAMD21 Dev Breakout
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Atmel SAM*: Atmel | SMART offers Flash- based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_samd21_dev_usb ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:sparkfun_samd21_dev_usb]
platform = atmelsam
board = sparkfun_samd21_dev_usb
```

You can override default SparkFun SAMD21 Dev Breakout settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_samd21_dev_usb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:sparkfun_samd21_dev_usb]
platform = atmelsam
board = sparkfun_samd21_dev_usb
```

(continues on next page)
Uploading

SparkFun SAMD21 Dev Breakout supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:sparkfun_samd21_dev_usb]
platform = atmelsam
board = sparkfun_samd21_dev_usb
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

SparkFun SAMD21 Dev Breakout does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun SAMD21 Mini Breakout

Contents

- SparkFun SAMD21 Mini Breakout
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Atmel SAM**: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use `sparkfun_samd21_mini_usb` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:sparkfun_samd21_mini_usb]
platform = atmelsam
board = sparkfun_samd21_mini_usb
```

You can override default SparkFun SAMD21 Mini Breakout settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sparkfun_samd21_mini_usb.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:sparkfun_samd21_mini_usb]
platform = atmelsam
board = sparkfun_samd21_mini_usb
```
Uploading

SparkFun SAMD21 Mini Breakout supports the next uploading protocols:
  - atmel-ice
  - blackmagic
  - jlink
  - sam-ba

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:sparkfun_samd21_mini_usb]
platform = atmelsam
board = sparkfun_samd21_mini_usb
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

SparkFun SAMD21 Mini Breakout does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun SAMD21 Pro RF

Contents

- SparkFun SAMD21 Pro RF
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_samd21_proRF ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:sparkfun_samd21_proRF]
platform = atmelsam
board = sparkfun_samd21_proRF
```

You can override default SparkFun SAMD21 Pro RF settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_samd21_proRF.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:sparkfun_samd21_proRF]
platform = atmelsam
board = sparkfun_samd21_proRF
```

(continues on next page)
Uploading

SparkFun SAMD21 Pro RF supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:sparkfun_samd21_proRF]
platform = atmelsam
board = sparkfun_samd21_proRF
upload_protocol = sam-ba
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun SAMD21 Pro RF does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SparkFun SAMD51 Thing Plus

Contents

- SparkFun SAMD51 Thing Plus
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD51J20A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>496KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_samd51_thing_plus ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_samd51_thing_plus]
platform = atmelsam
board = sparkfun_samd51_thing_plus
```

You can override default SparkFun SAMD51 Thing Plus settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_samd51_thing_plus.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sparkfun_samd51_thing_plus]
platform = atmelsam
board = sparkfun_samd51_thing_plus
```

(continues on next page)
; change microcontroller
board_build.mcu = samd51j20a

; change MCU frequency
board_build.f_cpu = 120000000L

### Uploading

SparkFun SAMD51 Thing Plus supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:sparkfun_samd51_thing_plus]
platform = atmelsam
board = sparkfun_samd51_thing_plus
upload_protocol = sam-ba
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *platformio.ini* (Project Configuration File).

SparkFun SAMD51 Thing Plus does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Tuino 096

Contents

- Tuino 096
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Atmel SAM: Atmel | SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>SAMD21G18A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Gimasi</td>
</tr>
</tbody>
</table>

Configuration

Please use tuinozero96 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:tuinozero96]
platform = atmelsam
board = tuinozero96
```

You can override default Tuino 096 settings per build environment using board_*** option, where *** is a JSON object path from board manifest tuinozero96.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:tuinozero96]
platform = atmelsam
board = tuinozero96

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Tuino 096 supports the next uploading protocols:
- atmel-ice
- blackmagic
- jlink
- stk500v2

Default protocol is stk500v2

You can change upload protocol using `upload_protocol` option:

```
[env:tuinozero96]
platform = atmel.sam
board = tuinozero96
upload_protocol = stk500v2
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Tuino 096 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel-ICE</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**1.12.6 CHIPS Alliance**

**RVfpga:** Digilent Nexys A7
• **RVfpga: Digilent Nexys A7**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **CHIPS Alliance**: The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>320MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1.16MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

### Configuration

Please use `swervolf_nexys` ID for `board` option in "`platformio.ini` (Project Configuration File):

```ini
[env:swervolf_nexys]
platform = chipsalliance
board = swervolf_nexys
```

You can override default RVfpga: Digilent Nexys A7 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `swervolf_nexys.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:swervolf_nexys]
platform = chipsalliance
board = swervolf_nexys

; change microcontroller
board_build.mcu =

; change MCU frequency
board_build.f_cpu = 320000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

RVfpga: Digilent Nexys A7 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG-HSI</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Verilator</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Whisper</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>WD-Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

**1.12.7 Espressif 32**

**AI Thinker ESP32-CAM**

Contents

- AI Thinker ESP32-CAM
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>AI Thinker</td>
</tr>
</tbody>
</table>

### Configuration

Please use `esp32cam` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:esp32cam]
platform = espressif32
board = esp32cam
```

You can override default AI Thinker ESP32-CAM settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `esp32cam.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32cam]
platform = espressif32
board = esp32cam

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### Uploading

AI Thinker ESP32-CAM supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:
[env:esp32cam]
platform = espressif32
board = esp32cam
upload_protocol = esptool

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

AI Thinker ESP32-CAM does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>Framework</td>
<td></td>
</tr>
</tbody>
</table>

ALKS ESP32

Contents

- ALKS ESP32
  - Hardware
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RoboticsBrno</td>
</tr>
</tbody>
</table>

Configuration

Please use `alksesp32` ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:alksesp32]
platform = espressif32
board = alksesp32
```

You can override default ALKS ESP32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `alksesp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:alksesp32]
platform = espressif32
board = alksesp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ALKS ESP32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:alksesp32]
platform = espressif32
board = alksesps32
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ALKS ESP32 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
AZ-Delivery ESP-32 Dev Kit C V4

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>520KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>AZ-Delivery</td>
</tr>
</tbody>
</table>

Configuration

Please use `az-delivery-devkit-v4` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:az-delivery-devkit-v4]
platform = espressif32
board = az-delivery-devkit-v4
```

You can override default AZ-Delivery ESP-32 Dev Kit C V4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `az-delivery-devkit-v4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:az-delivery-devkit-v4]
platform = espressif32
board = az-delivery-devkit-v4

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

AZ-Delivery ESP-32 Dev Kit C V4 supports the next uploading protocols:
platformio documentation, release 5.0.5a1

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:az-delivery-devkit-v4]
platform = espressif32
board = az-delivery-devkit-v4
upload_protocol = esptool
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

AZ-Delivery ESP-32 Dev Kit C V4 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.12. Boards
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>Framework</td>
<td></td>
</tr>
</tbody>
</table>

Adafruit ESP32 Feather

Contents

- Adafruit ESP32 Feather
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `featheresp32` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:featheresp32]
platform = espressif32
board = featheresp32
```

You can override default Adafruit ESP32 Feather settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `featheresp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:featheresp32]
platform = espressif32
board = featheresp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L

### Uploading

Adafruit ESP32 Feather supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

[env:featheresp32]
platform = espressif32
board = featheresp32
upload_protocol = esptool

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).
Adafruit ESP32 Feather does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**April Brother ESPea32**

**Contents**

- April Brother ESPea32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>April Brother</td>
</tr>
</tbody>
</table>
Configuration

Please use `espea32` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:espea32]
platform = espressif32
board = espea32
```

You can override default April Brother ESPea32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `espea32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:espea32]
platform = espressif32
board = espea32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

April Brother ESPea32 supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:espea32]
platform = espressif32
board = espea32

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support April Brother ESPea32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
BPI-Bit

Contents

- BPI-Bit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>160MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BPI Tech</td>
</tr>
</tbody>
</table>

Configuration

Please use *bpi-bit* ID for *board* option in *“platformio.ini” (Project Configuration File)*:

```ini
[env:bpi-bit]
platform = espressif32
board = bpi-bit
```

You can override default BPI-Bit settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *bpi-bit.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```ini
[env:bpi-bit]
platform = espressif32
board = bpi-bit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 160000000L
```

Uploading

BPI-Bit supports the next uploading protocols:
• espota
• esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:bpi-bit]
platform = espressif32
board = bpi-bit
upload_protocol = esptool
```

### Debugging

Debugging currently does not support BPI-Bit board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### Briki ABC (MBC-WB) - ESP32

#### Contents

- Briki ABC (MBC-WB) - ESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
Microcontroller | ESP32  
--- | ---  
Frequency | 240MHz  
Flash | 3.25MB  
RAM | 320KB  
Vendor | meteca  

## Configuration

Please use briki_abc_esp32 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:briki_abc_esp32]
platform = espressif32
board = briki_abc_esp32
```

You can override default Briki ABC (MBC-WB) - ESP32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest briki_abc_esp32.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:briki_abc_esp32]
platform = espressif32
board = briki_abc_esp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

## Uploading

Briki ABC (MBC-WB) - ESP32 supports the next uploading protocols:

- esp-prog
- iot-bus-jtag
- jlink
- mbctool
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is mbctool

You can change upload protocol using `upload_protocol` option:
[env:briki_abc.esp32]
platform = espressif32
board = briki_abc.esp32
upload_protocol = mbctool

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in *“platformio.ini” (Project Configuration File)*.

Briki ABC (MBC-WB) - ESP32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Briki MBC-WB - ESP32

Contents

- *Briki MBC-WB - ESP32*
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>3.25MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>meteca</td>
</tr>
</tbody>
</table>

Configuration

Please use `briki_mbc-wb_esp32` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:briki_mbc-wb_esp32]
platform = espressif32
board = briki_mbc-wb_esp32
```

You can override default Briki MBC-WB - ESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `briki_mbc-wb_esp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:briki_mbc-wb_esp32]
platform = espressif32
board = briki_mbc-wb_esp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Briki MBC-WB - ESP32 supports the next uploading protocols:

- esp-prog
- iot-bus-jtag
- jlink
- mbc
- mbctool
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is mbctool

You can change upload protocol using `upload_protocol` option:

```
[env:briki_mbc-wb_esp32]
platform = espressif32
board = briki_mbc-wb_esp32
upload_protocol = mbctool
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (**Project Configuration File**).

Briki MBC-WB - ESP32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**D-duino-32**
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>DSTIKE</td>
</tr>
</tbody>
</table>

Configuration

Please use `d-duino-32` ID for `board` option in “`platformio.ini`” (*Project Configuration File)*:

```
[env:d-duino-32]
platform = espressif32
board = d-duino-32
```

You can override default D-duino-32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `d-duino-32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:d-duino-32]
platform = espressif32
board = d-duino-32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

D-duino-32 supports the next uploading protocols:

- esp-prog
- espota
• esptool
• iot-bus-jtag
• jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:d-duino-32]
platform = espressif32
board = d-duino-32
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

D-duino-32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESPRESSIF IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### DOIT ESP32 DEVKIT V1

#### Contents

- **DOIT ESP32 DEVKIT V1**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>DOIT</td>
</tr>
</tbody>
</table>

### Configuration

Please use `esp32doit-devkit-v1` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:esp32doit-devkit-v1]
platform = esp32f32
board = esp32doit-devkit-v1
```

You can override default DOIT ESP32 DEVKIT V1 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `esp32doit-devkit-v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

DOIT ESP32 DEVKIT V1 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.
DOIT ESP32 DEVKIT V1 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**Dongsen Tech Pocket 32**

**Contents**

- **Dongsen Tech Pocket 32**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Dongsen Technology</td>
</tr>
</tbody>
</table>
Configuration

Please use `pocket_32` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:pocket_32]
platform = espressif32
board = pocket_32
```

You can override default Dongsen Tech Pocket 32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `pocket_32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:pocket_32]
platform = espressif32
board = pocket_32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Dongsen Tech Pocket 32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:pocket_32]
platform = espressif32
board = pocket_32

upload_protocol = esptool
```
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Dongsen Tech Pocket 32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**ESP32 FM DevKit**

**Contents**

- ESP32 FM DevKit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Configuration

Please use `fm-devkit` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:fm-devkit]
platform = espressif32
board = fm-devkit
```

You can override default ESP32 FM DevKit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `fm-devkit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:fm-devkit]
platform = espressif32
board = fm-devkit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ESP32 FM DevKit supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-oct
- olimex-arm-usb-oct-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
• tumpa

Default protocol is esptool
You can change upload protocol using `upload_protocol` option:

```
[env:fm-devkit]
platform = espressif32
board = fm-devkit
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ESP32 FM DevKit does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**ESP32 Pico Kit**
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use `pico32` ID for `board` option in "`platformio.ini` (Project Configuration File):

```
[env:pico32]
platform = espressif32
board = pico32
```

You can override default ESP32 Pico Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `pico32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pico32]
platform = espressif32
board = pico32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ESP32 Pico Kit supports the next uploading protocols:

- `esp-prog`
- `espota`
• esptool
• iot-bus-jtag
• jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is **esptool**

You can change upload protocol using `upload_protocol` option:

```
[env:pico32]
platform = espressif32
board = pico32
upload_protocol = esptool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ESP32 Pico Kit does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><em>oddWires IOT-Bus JTAG</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Mini-Module FT2232H</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-OCD</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-OCD-H</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-TINY-H</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-TINY</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>TIAO USB Multi-Protocol Adapter (TUMPA)</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP32vn IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

ESP32vn IoT Uno

Contents

- ESP32vn IoT Uno
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ESP32vn</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp32vn-iot-uno` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:esp32vn-iot-uno]
platform = espressif32
board = esp32vn-iot-uno
```

You can override default ESP32vn IoT Uno settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `esp32vn-iot-uno.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.
[env:esp32vn-iot-uno]
platform = espressif32
board = esp32vn-iot-uno

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L

### Uploading

ESP32vn IoT Uno supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

[env:esp32vn-iot-uno]
platform = espressif32
board = esp32vn-iot-uno
upload_protocol = esptool

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).
ESP32vn IoT Uno does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### ESPectro32

**Contents**

- ESPectro32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>DycodeX</td>
</tr>
</tbody>
</table>
Configuration

Please use espectro32 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:espectro32]
platform = espressif32
board = espectro32
```

You can override default ESPectro32 settings per build environment using board_*** option, where *** is a JSON object path from board manifest espectro32.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:espectro32]
platform = espressif32
board = espectro32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ESPectro32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:espectro32]
platform = espressif32
board = espectro32

upload_protocol = esptool
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ESPectro32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP-IDF</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

ESPino32

Contents

- ESPino32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ThaiEasyElec</td>
</tr>
</tbody>
</table>

Configuration

Please use espino32 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:espino32]
platform = espressif32
board = espino32
```

You can override default ESPino32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest espino32.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:espino32]
platform = espressif32
board = espino32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ESPino32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:espino32]
platform = espressif32
board = espino32
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File). ESPino32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESP-Prog</strong></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><em>oddWires IOT-Bus JTAG</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>J-LINK</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Mini-Module FT2232H</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-OCD</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-OCD-H</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-TINY-H</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Olimex ARM-USB-TINY</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>TIAO USB Multi-Protocol Adapter (TUMPA)</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><em>Espressif IoT Development Framework</em></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**Electronic SweetPeas ESP320**
Contents

- Electronic SweetPeas ESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Electronic SweetPeas</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp320` ID for `board` option in “platformio.ini” (*Project Configuration File)*:

```
[env:esp320]
platform = espressif32
board = esp320
```

You can override default Electronic SweetPeas ESP320 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp320.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp320]
platform = espressif32
board = esp320

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Electronic SweetPeas ESP320 supports the next uploading protocols:

- espota
- esptool
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```python
[env:esp320]
platform = espressif32
board = esp320
upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support Electronic SweetPeas ESP320 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### Espressif ESP-WROVER-KIT

#### Contents

- `Espressif ESP-WROVER-KIT`
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform `Espressif 32`: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>
Configuration

Please use `esp-wrover-kit` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:esp-wrover-kit]
platform = espressif32
board = esp-wrover-kit
```

You can override default Espressif ESP-WROVER-KIT settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp-wrover-kit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:esp-wrover-kit]
platform = espressif32
board = esp-wrover-kit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Espressif ESP-WROVER-KIT supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `ftdi`
- `iot-bus-jtag`
- `jlink`
- `minimodule`
- `olimex-arm-usb-ocd`
- `olimex-arm-usb-ocd-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
- `tumpa`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:esp-wrover-kit]
platform = espressif32
board = esp-wrover-kit

upload_protocol = esptool
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Espressif ESP-WROVER-KIT has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Espressif ESP32 Dev Module

Contents

- **Espressif ESP32 Dev Module**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
PlatformIO Documentation, Release 5.0.5a1

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp32dev` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```
[env:esp32dev]
platform = espressif32
board = esp32dev
```

You can override default Espressif ESP32 Dev Module settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `esp32dev.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32dev]
platform = espressif32
board = esp32dev

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Espressif ESP32 Dev Module supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `iot-bus-jtag`
- `jlink`
- `minimodule`
- `olimex-arm-usb-ocd`
- `olimex-arm-usb-ocd-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
• tumpa

Default protocol is **esptool**

You can change upload protocol using *upload_protocol* option:

```
[env:esp32dev]
platform = espressif32
board = esp32dev
upload_protocol = esptool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Espressif ESP32 Dev Module does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT</strong> Development Framework</td>
<td><strong>ESP-IDF</strong> is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**FireBeetle-ESP32**
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>DFRobot</td>
</tr>
</tbody>
</table>

Configuration

Please use `firebeetle32` ID for *board* option in "platformio.ini" (Project Configuration File):

```
[env:firebeetle32]
platform = espressif32
board = firebeetle32
```

You can override default FireBeetle-ESP32 settings per build environment using *board_*** option, where *** is a JSON object path from board manifest `firebeetle32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:firebeetle32]
platform = espressif32
board = firebeetle32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

FireBeetle-ESP32 supports the next uploading protocols:

- esp-prog
- espota
• esptool
• iot-bus-jtag
• jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:firebeetle32]
platform = espressif32
board = firebeetle32
upload_protocol = esptool
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” *(Project Configuration File)*.

FireBeetle-ESP32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Frog Board ESP32

Contents

- Frog Board ESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Fred</td>
</tr>
</tbody>
</table>

Configuration

Please use frogboard ID for board option in "platformio.ini" (Project Configuration File):

```
[env:frogboard]
platform = espressif32
board = frogboard
```

You can override default Frog Board ESP32 settings per build environment using board_*** option, where *** is a JSON object path from board manifest frogboard.json. For example, board_build.mcu, board_build.f_cpu, etc.
Frog Board ESP32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:frogboard]
platform = espressif32
board = frogboard

upload_protocol = esptool
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.
Frog Board ESP32 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Heltec WiFi Kit 32

Contents

- Heltec WiFi Kit 32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec Automation</td>
</tr>
</tbody>
</table>
Configuration

Please use heltec_wifi_kit_32 ID for `board` option in “platformio.ini” (Project Configuration File):

```python
[env:heltec_wifi_kit_32]
platform = espressif32
board = heltec_wifi_kit_32
```

You can override default Heltec WiFi Kit 32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest heltec_wifi_kit_32.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:heltec_wifi_kit_32]
platform = espressif32
board = heltec_wifi_kit_32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Heltec WiFi Kit 32 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```python
[env:heltec_wifi_kit_32]
platform = espressif32
board = heltec_wifi_kit_32

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support Heltec WiFi Kit 32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
Heltec WiFi LoRa 32

Contents

- Heltec WiFi LoRa 32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec Automation</td>
</tr>
</tbody>
</table>

Configuration

Please use *heltec_wifi_lora_32* ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:heltec_wifi_lora_32]
platform = espressif32
board = heltec_wifi_lora_32
```

You can override default Heltec WiFi LoRa 32 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *heltec_wifi_lora_32.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```ini
[env:heltec_wifi_lora_32]
platform = espressif32
board = heltec_wifi_lora_32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Heltec WiFi LoRa 32 supports the next uploading protocols:
- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:heltec_wifi_lora_32]
platform = espressif32
board = heltec_wifi_lora_32

upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

Heltec WiFi LoRa 32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Heltec WiFi LoRa 32 (V2)

Contents

- Heltec WiFi LoRa 32 (V2)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec Automation</td>
</tr>
</tbody>
</table>

Configuration

Please use `heltec_wifi_lora_32_V2` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:heltec_wifi_lora_32_V2]
platform = espressif32
board = heltec_wifi_lora_32_V2
```

You can override default Heltec WiFi LoRa 32 (V2) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `heltec_wifi_lora_32_V2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:heltec_wifi_lora_32_V2]
platform = espressif32
board = heltec_wifi_lora_32_V2

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L

Uploading

Heltec WiFi LoRa 32 (V2) supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-oct
- olimex-arm-usb-oct-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

[env:heltec_wifi_lora_32_V2]
platform = espressif32
board = heltec_wifi_lora_32_V2

upload_protocol = esptool

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
Heltec WiFi LoRa 32 (V2) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Heltec Wireless Stick

Contents

- Heltec Wireless Stick
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Heltec Automation</td>
</tr>
</tbody>
</table>
Configuration

Please use `heltec_wireless_stick` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:heltec_wireless_stick]
platform = espressif32
board = heltec_wireless_stick
```

You can override default Heltec Wireless Stick settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `heltec_wireless_stick.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:heltec_wireless_stick]
platform = espressif32
board = heltec_wireless_stick

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Heltec Wireless Stick supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-oct-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:heltec_wireless_stick]
platform = espressif32
board = heltec_wireless_stick

upload_protocol = esptool
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Heltec Wireless Stick does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Hornbill ESP32 Dev

**Contents**

- **Hornbill ESP32 Dev**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Hornbill</td>
</tr>
</tbody>
</table>

Configuration

Please use hornbill32dev ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:hornbill32dev]
platform = espressif32
board = hornbill32dev
```

You can override default Hornbill ESP32 Dev settings per build environment using `board_***` option, where *** is a JSON object path from board manifest hornbill32dev.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:hornbill32dev]
platform = espressif32
board = hornbill32dev

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Hornbill ESP32 Dev supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
• tumpa

Default protocol is **esptool**

You can change upload protocol using `upload_protocol` option:

```
[env:hornbill32dev]
platform = espressif32
board = hornbill32dev
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Hornbill ESP32 Dev does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**Hornbill ESP32 Minima**
Contents

- Hornbill ESP32 Minima
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Hornbill</td>
</tr>
</tbody>
</table>

Configuration

Please use hornbill32minima ID for board option in “platformio.ini” (Project Configuration File):

```
[env:hornbill32minima]
platform = espressif32
board = hornbill32minima
```

You can override default Hornbill ESP32 Minima settings per build environment using board_*** option, where *** is a JSON object path from board manifest hornbill32minima.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:hornbill32minima]
platform = espressif32
board = hornbill32minima

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Hornbill ESP32 Minima supports the next uploading protocols:

- esp-prog
- espota
• esptool
• iot-bus-jtag
• jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool
You can change upload protocol using upload_protocol option:

```
[env:hornbill32minima]
platform = espressif32
board = hornbill32minima
upload_protocol = esptool
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Hornbill ESP32 Minima does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

IntoRobot Fig

Contents

- IntoRobot Fig
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>IntoRobot</td>
</tr>
</tbody>
</table>

Configuration

Please use `intorobot ID` for `board` option in "platformio.ini" (Project Configuration File):

```conf
[env:intorobot]
platform = espressif32
board = intorobot
```

You can override default IntoRobot Fig settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `intorobot.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.
### Uploading

IntoRobot Fig supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```yaml
[env:intorobot]
platform = espressif32
board = intorobot
upload_protocol = esptool
```

### Debugging

*Debugging* currently does not support IntoRobot Fig board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### IoTaaP Magnolia

- **IoTaaP Magnolia**
  - Hardware
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>IoTaaP</td>
</tr>
</tbody>
</table>

Configuration

Please use `iotaap_magnolia` ID for *board* option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:iotaap_magnolia]
platform = espressif32
board = iotaap_magnolia
```

You can override default IoTaaP Magnolia settings per build environment using *board_**** option, where *** is a JSON object path from board manifest `iotaap_magnolia.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:iotaap_magnolia]
platform = espressif32
board = iotaap_magnolia

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

IoTaaP Magnolia supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
• minimodule
• olimex-arm-usb-oct
• olimex-arm-usb-oct-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:iotaap_magnolia]
platform = espressif32
board = iotaap_magnolia
upload_protocol = esptool
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

IoTaaP Magnolia does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
M5Stack Core ESP32

Contents

- M5Stack Core ESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>M5Stack</td>
</tr>
</tbody>
</table>

Configuration

Please use m5stack-core-esp32 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32
```

You can override default M5Stack Core ESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest m5stack-core-esp32.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

M5Stack Core ESP32 supports the next uploading protocols:
•.espota
•.esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32
upload_protocol = esptool
```

### Debugging

**Debugging** currently does not support M5Stack Core ESP32 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### M5Stack FIRE

#### Contents

- M5Stack FIRE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
**Configuration**

Please use `m5stack-fire` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:m5stack-fire]
platform = espressif32
board = m5stack-fire
```

You can override default M5Stack FIRE settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `m5stack-fire.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.:

```ini
[env:m5stack-fire]
platform = espressif32
board = m5stack-fire

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

**Uploading**

M5Stack FIRE supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:m5stack-fire]
platform = espressif32
board = m5stack-fire

upload_protocol = esptool
```

**Debugging**

`Debugging` currently does not support M5Stack FIRE board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

M5Stack GREY ESP32

Contents

- M5Stack GREY ESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>520KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>M5Stack</td>
</tr>
</tbody>
</table>

Configuration

Please use m5stack-grey ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:m5stack-grey]
platform = espressif32
board = m5stack-grey
```

You can override default M5Stack GREY ESP32 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest m5stack-grey.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:m5stack-grey]
platform = espressif32
board = m5stack-grey

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L

Uploading

M5Stack GREY ESP32 supports the next uploading protocols:

• espota
• esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

[env:m5stack-grey]
platform = espressif32
board = m5stack-grey

upload_protocol = esptool

Debugging

Debugging currently does not support M5Stack GREY ESP32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

M5Stick-C

Contents

• M5Stick-C
  - Hardware
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>M5Stack</td>
</tr>
</tbody>
</table>

Configuration

Please use `m5stick-c` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:m5stick-c]
platform = espressif32
board = m5stick-c
```

You can override default M5Stick-C settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `m5stick-c.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:m5stick-c]
platform = espressif32
board = m5stick-c

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

M5Stick-C supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:
[[env:m5stick-c]]
```yaml
platform = espressif32
board = m5stick-c
upload_protocol = esptool
```

### Debugging

*Debugging* currently does not support M5Stick-C board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### MH ET LIVE ESP32DevKIT

#### Contents

- MH ET LIVE ESP32DevKIT
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MH-ET Live</td>
</tr>
</tbody>
</table>
Configuration

Please use `mhetesp32devkit` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:mhetesp32devkit]
platform = espressif32
board = mhetesp32devkit
```

You can override default MH ET LIVE ESP32DevKIT settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mhetesp32devkit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mhetesp32devkit]
platform = espressif32
board = mhetesp32devkit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

MH ET LIVE ESP32DevKIT supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:mhetesp32devkit]
platform = espressif32
board = mhetesp32devkit

upload_protocol = esptool
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

MH ET LIVE ESP32DevKIT does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif Iot Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

MH ET LIVE ESP32MiniKit

- MH ET LIVE ESP32MiniKit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MH-ET Live</td>
</tr>
</tbody>
</table>

Configuration

Please use `mhetesp32minikit` ID for *board* option in “`platformio.ini` (Project Configuration File):

```ini
[env:mhetesp32minikit]
platform = espressif32
board = mhetesp32minikit
```

You can override default MH ET LIVE ESP32MiniKit settings per build environment using `board_*` option, where *** is a JSON object path from board manifest `mhetesp32minikit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mhetesp32minikit]
platform = espressif32
board = mhetesp32minikit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

MH ET LIVE ESP32MiniKit supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `iot-bus-jtag`
- `jlink`
- `minimodule`
- `olimex-arm-usb-oct`
- `olimex-arm-usb-oct-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
• tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:mhetesp32minikit]
platform = espressif32
board = mhetesp32minikit
upload_protocol = esptool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” *(Project Configuration File)*.

MH ET LIVE ESP32MiniKit does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### MagicBit

1297
Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Magicblocks.io</td>
</tr>
</tbody>
</table>

Configuration

Please use `magicbit ID` for `board` option in “platformio.ini” (Project Configuration File):

```
[env:magicbit]
platform = espressif32
board = magicbit
```

You can override default MagicBit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `magicbit.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:magicbit]
platform = espressif32
board = magicbit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

MagicBit supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:magicbit]
platform = espressif32
board = magicbit

upload_protocol = esptool
```

**Debugging**

Debugging currently does not support MagicBit board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>Framework</td>
<td></td>
</tr>
</tbody>
</table>

**MakerAsia Nano32**

**Contents**

- MakerAsia Nano32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MakerAsia</td>
</tr>
</tbody>
</table>
Configuration

Please use nano32 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nano32]
platform = espressif32
board = nano32
```

You can override default MakerAsia Nano32 settings per build environment using board_*** option, where *** is a JSON object path from board manifest nano32.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nano32]
platform = espressif32
board = nano32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

MakerAsia Nano32 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```ini
[env:nano32]
platform = espressif32
board = nano32

upload_protocol = esptool
```

Debugging

Debugging currently does not support MakerAsia Nano32 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>Pumbaa</td>
<td>Pumbaa is Python on top of Simba. The implementation is a port of MicroPython, designed for embedded devices with limited amount of RAM and code memory</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

Microduino Core ESP32

Contents

- Microduino Core ESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `microduino-core-esp32` ID for `board` option in “platformio.ini” (Project Configuration File):

```plaintext
[env:microduino-core-esp32]
platform = esp32
board = microduino-core-esp32
```
You can override default Microduino Core ESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `microduino-core-esp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:microduino-core-esp32]
platform = espressif32
board = microduino-core-esp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### Uploading

Microduino Core ESP32 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:microduino-core-esp32]
platform = espressif32
board = microduino-core-esp32

upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support Microduino Core ESP32 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### Node32s

<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
</table>
Node32s

- Hardware
- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Aiyarafun</td>
</tr>
</tbody>
</table>

Configuration

Please use `node32s` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:node32s]
platform = espressif32
board = node32s
```

You can override default Node32s settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `node32s.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:node32s]
platform = espressif32
board = node32s

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Node32s supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
• jlink
• minimodule
• olimex-arm-usb-oct
• olimex-arm-usb-oct-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:node32s]
platform = espressif32
board = node32s
upload_protocol = esptool
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Node32s does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

NodeMCU-32S

Contents

- NodeMCU-32S
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NodeMCU</td>
</tr>
</tbody>
</table>

Configuration

Please use `nodemcu-32s` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s
```

You can override default NodeMCU-32S settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nodemcu-32s.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

**Uploading**

NodeMCU-32S supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `iot-bus-jtag`
- `jlink`
- `minimodule`
- `olimex-arm-usb-ocd`
- `olimex-arm-usb-ocd-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
- `tumpa`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s

upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File).*
NodeMCU-32S does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESP-Prog</strong></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>oddWires IOT-Bus JTAG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>J-LINK</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mini-Module FT2232H</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Olimex ARM-USB-OCD</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Olimex ARM-USB-OCD-H</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Olimex ARM-USB-TINY-H</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Olimex ARM-USB-TINY</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TIAO USB Multi-Protocol Adapter (TUMPA)</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### Noduino Quantum

#### Contents

- **Noduino Quantum**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Noduino</td>
</tr>
</tbody>
</table>
Configuration

Please use `quantum` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:quantum]
platform = espressif32
board = quantum
```

You can override default Noduino Quantum settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `quantum.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:quantum]
platform = espressif32
board = quantum

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Noduino Quantum supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:quantum]
platform = espressif32
board = quantum

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support Noduino Quantum board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
ODROID-GO

Contents

- ODROID-GO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Hardkernel</td>
</tr>
</tbody>
</table>

Configuration

Please use odroid_esp32 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:odroid_esp32]
platform = espressif32
board = odroid_esp32
```

You can override default ODROID-GO settings per build environment using board_*** option, where *** is a JSON object path from board manifest odroid_esp32.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:odroid_esp32]
platform = espressif32
board = odroid_esp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ODROID-GO supports the next uploading protocols:
• espota
• esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:odroid_esp32]
platform = espressif32
board = odroid_esp32
upload_protocol = esptool
```

Debugging

Debugging currently does not support ODROID-GO board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

OLIMEX ESP32-DevKit-LiPo

Contents

• OLIMEX ESP32-DevKit-LiPo
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
## Configuration

Please use `esp32-devkitlipo` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:esp32-devkitlipo]
platform = espressif32
board = esp32-devkitlipo
```

You can override default OLIMEX ESP32-DevKit-LiPo settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `esp32-devkitlipo.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:esp32-devkitlipo]
platform = espressif32
board = esp32-devkitlipo

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

## Uploading

OLIMEX ESP32-DevKit-LiPo supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:
[env:esp32-devkitlipo]
platform = espressif32
board = esp32-devkitlipo
upload_protocol = esptool

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

OLIMEX ESP32-DevKit-LiPo does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif</strong> IoT Development <strong>Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

OLIMEX ESP32-EVB

Contents

- OLIMEX ESP32-EVB
  - Hardware
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>OLIMEX</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp32-evb` ID for `board` option in “platformio.ini” (*Project Configuration File*):

```
[env:esp32-evb]
platform = espressif32
board = esp32-evb
```

You can override default OLIMEX ESP32-EVB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32-evb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32-evb]
platform = espressif32
board = esp32-evb

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

OLIMEX ESP32-EVB supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
• minimodule
• olimex-arm-usb-oecd
• olimex-arm-usb-oecd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:esp32-evb]
platform = espressif32
board = esp32-evb

upload_protocol = esptool
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using debug_tool option in “platformio.ini” (Project Configuration File).

OLIMEX ESP32-EVB does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

OLIMEX ESP32-GATEWAY

Contents

- OLIMEX ESP32-GATEWAY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>OLIMEX</td>
</tr>
</tbody>
</table>

Configuration

Please use esp32-gateway ID for board option in “platformio.ini” (Project Configuration File):

```
[env:esp32-gateway]
platform = espressif32
board = esp32-gateway
```

You can override default OLIMEX ESP32-GATEWAY settings per build environment using board_*** option, where *** is a JSON object path from board manifest esp32-gateway.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

OLIMEX ESP32-GATEWAY supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:esp32-gateway]
platform = espressif32
board = esp32-gateway

upload_protocol = esptool
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).
OLIMEX ESP32-GATEWAY does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

## OLIMEX ESP32-PRO

### Contents

- **OLIMEX ESP32-PRO**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>OLIMEX</td>
</tr>
</tbody>
</table>
Configuration

Please use esp32-pro ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:esp32-pro]
platform = esp32
board = esp32-pro
```

You can override default OLIMEX ESP32-PRO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32-pro.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:esp32-pro]
platform = esp32
board = esp32-pro

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

OLIMEX ESP32-PRO supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:esp32-pro]
platform = esp32
board = esp32-pro

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support OLIMEX ESP32-PRO board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
OLIMEX ESP32-PoE

Contents

- OLIMEX ESP32-PoE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>OLIMEX</td>
</tr>
</tbody>
</table>

Configuration

Please use esp32-poe ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:esp32-poe]
platform = espressif32
board = esp32-poe
```

You can override default OLIMEX ESP32-PoE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32-poe.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:esp32-poe]
platform = espressif32
board = esp32-poe

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

OLIMEX ESP32-PoE supports the next uploading protocols:
• espota
• esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp32-poe]
platform = espressif32
board = esp32-poe
upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support OLIMEX ESP32-PoE board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP-IDF Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### OLIMEX ESP32-PoE-ISO

#### Contents

- OLIMEX ESP32-PoE-ISO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
### Configuration

Please use `esp32-poe-iso` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:esp32-poe-iso]
platform = espressif32
board = esp32-poe-iso
```

You can override default OLIMEX ESP32-PoE-ISO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32-poe-iso.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32-poe-iso]
platform = espressif32
board = esp32-poe-iso

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### Uploading

OLIMEX ESP32-PoE-ISO supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp32-poe-iso]
platform = espressif32
board = esp32-poe-iso

upload_protocol = esptool
```

### Debugging

*Debugging* currently does not support OLIMEX ESP32-PoE-ISO board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

OROCA EduBot

Contents

- **OROCA EduBot**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>OROCA</td>
</tr>
</tbody>
</table>

Configuration

Please use `oroca_edubot` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:oroca_edubot]
platform = espressif32
board = oroca_edubot
```

You can override default OROCA EduBot settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `oroca_edubot.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`
Uploading

OROCA EduBot supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```yaml
[env:oroca_edubot]
platform = espressif32
board = oroca_edubot

upload_protocol = esptool
```

Debugging

*Debugging* currently does not support OROCA EduBot board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Onehorse ESP32 Dev Module

Contents

- Onehorse ESP32 Dev Module
  - Hardware
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Onehorse</td>
</tr>
</tbody>
</table>

Configuration

Please use `onehorse32dev` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:onehorse32dev]
platform = espressif32
board = onehorse32dev
```

You can override default Onehorse ESP32 Dev Module settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `onehorse32dev.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:onehorse32dev]
platform = espressif32
board = onehorse32dev

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Onehorse ESP32 Dev Module supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:
Debugging

Debugging currently does not support Onehorse ESP32 Dev Module board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Pycom GPy

Contents

- Pycom GPy
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Pycom Ltd.</td>
</tr>
</tbody>
</table>
Configuration

Please use `pycom_gpy` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:pycom_gpy]
platform = espressif32
board = pycom_gpy
```

You can override default Pycom GPy settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `pycom_gpy.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pycom_gpy]
platform = espressif32
board = pycom_gpy

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Pycom GPy supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:pycom_gpy]
platform = espressif32
board = pycom_gpy

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support Pycom GPy board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espresif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
Pycom LoPy

Contents
- Pycom LoPy
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Pycom Ltd.</td>
</tr>
</tbody>
</table>

Configuration

Please use `lopy` ID for `board` option in “platformio.ini” (*Project Configuration File)*:

```plaintext
[env:lopy]
platform = espressif32
board = lopy
```

You can override default Pycom LoPy settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lopy.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:lopy]
platform = espressif32
board = lopy

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Pycom LoPy supports the next uploading protocols:
• esp-prog
• espota
• esptool
• iot-bus-jtag
• jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:lopy]
platform = espressif32
board = lopy
upload_protocol = esptool
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Pycom LoPy does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Pycom LoPy4

Contents

- Pycom LoPy4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1.25MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Pycom Ltd.</td>
</tr>
</tbody>
</table>

Configuration

Please use `lopy4` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:lopy4]
platform = espressif32
board = lopy4
```

You can override default Pycom LoPy4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lopy4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

Pycom LoPy4 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```ini
[env:lopy4]
platform = espressif32
board = lopy4
upload_protocol = esptool
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” *(Project Configuration File)*.
Pycom LoPy4 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**Qchip**

**Contents**

- **Qchip**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Qmobot LLP</td>
</tr>
</tbody>
</table>
Configuration

Please use `qchip` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:qchip]
platform = espressif32
board = qchip
```

You can override default Qchip settings per build environment using `board/***` option, where `***` is a JSON object path from board manifest `qchip.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:qchip]
platform = espressif32
board = qchip

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Qchip supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:qchip]
platform = espressif32
board = qchip

upload_protocol = esptool
```

Debugging

*Debugging* currently does not support Qchip board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
SG-O AirMon

Contents

- SG-O AirMon
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SG-O</td>
</tr>
</tbody>
</table>

Configuration

Please use sg-o_airMon ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:sg-o_airMon]
platform = espressif32
board = sg-o_airMon
```

You can override default SG-O AirMon settings per build environment using board_*** option, where *** is a JSON object path from board manifest sg-o_airMon.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:sg-o_airMon]
platform = espressif32
board = sg-o_airMon

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

SG-O AirMon supports the next uploading protocols:
- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:sg-o_airMon]
platform = espressif32
board = sg-o_airMon
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” *(Project Configuration File)*.

SG-O AirMon does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Silicognition wESP32

Contents

- Silicognition wESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32:* Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicognition</td>
</tr>
</tbody>
</table>

Configuration

Please use `wesp32` ID for `board` option in “`platformio.ini`” *(Project Configuration File):*

```
[env:wesp32]
platform = espressif32
board = wesp32
```

You can override default Silicognition wESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wesp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

Silicognition wESP32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (*Project Configuration File*).
Silicognition wESP32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

### SparkFun ESP32 Thing

#### Contents

- SparkFun ESP32 Thing
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun Electronics</td>
</tr>
</tbody>
</table>
Configuration

Please use esp32thing ID for board option in “platformio.ini” (Project Configuration File):

```
[env:esp32thing]
platform = espressif32
board = esp32thing
```

You can override default SparkFun ESP32 Thing settings per build environment using board_*** option, where *** is a JSON object path from board manifest esp32thing.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:esp32thing]
platform = espressif32
board = esp32thing

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

SparkFun ESP32 Thing supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:esp32thing]
platform = espressif32
board = esp32thing

upload_protocol = esptool
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun ESP32 Thing does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

SparkFun LoRa Gateway 1-Channel

Contents

- SparkFun LoRa Gateway 1-Channel
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use `sparkfun_lora_gateway_1-channel ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_lora_gateway_1-channel]
platform = espressif32
board = sparkfun_lora_gateway_1-channel
```

You can override default SparkFun LoRa Gateway 1-Channel settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_lora_gateway_1-channel.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparkfun_lora_gateway_1-channel]
platform = espressif32
board = sparkfun_lora_gateway_1-channel

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

SparkFun LoRa Gateway 1-Channel supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
• **tumpa**

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:sparkfun_lora_gateway_1-channel]
platform = espressif32
board = sparkfun_lora_gateway_1-channel
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

SparkFun LoRa Gateway 1-Channel does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**TTGO LoRa32-OLED V1**
Contents

- TTGO LoRa32-OLED V1
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TTGO</td>
</tr>
</tbody>
</table>

Configuration

Please use ttgo-lora32-v1 ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:ttgo-lora32-v1]
platform = espressif32
board = ttgo-lora32-v1
```

You can override default TTGO LoRa32-OLED V1 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest ttgo-lora32-v1.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ttgo-lora32-v1]
platform = espressif32
board = ttgo-lora32-v1

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

TTGO LoRa32-OLED V1 supports the next uploading protocols:

- esp-prog
- espota
• esptool
• iot-bus-jtag
• jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:ttgo-lora32-v1]
platform = espressif32
board = ttgo-lora32-v1
upload_protocol = esptool
```

## Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

TTGO LoRa32-OLED V1 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**TTGO LoRa32-OLED V2**

**Contents**

- TTGO LoRa32-OLED V2
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TTGO</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `ttgo-lora32-v2` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ttgo-lora32-v2]
platform = espressif32
board = ttgo-lora32-v2
```

You can override default TTGO LoRa32-OLED V2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `ttgo-lora32-v2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:ttgo-lora32-v2]
platform = espressif32
board = ttgo-lora32-v2

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L

Uploading

TTGO LoRa32-OLED V2 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

[env:ttgo-lora32-v2]
platform = espressif32
board = ttgo-lora32-v2
upload_protocol = esptool

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
TTGO LoRa32-OLED V2 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

TTGO T-Beam

Contents

- TTGO T-Beam
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
</tbody>
</table>
### Configuration

Please use `ttgo-t-beam` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:ttgo-t-beam]
platform = espressif32
board = ttgo-t-beam
```

You can override default TTGO T-Beam settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ttgo-t-beam.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ttgo-t-beam]
platform = espressif32
board = ttgo-t-beam

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### Uploading

TTGO T-Beam supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:ttgo-t-beam]
platform = espressif32
board = ttgo-t-beam

upload_protocol = esptool
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.

TTGO T-Beam does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Espressif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

TTGO T-Watch

**Contents**

- **TTGO T-Watch**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TTGO</td>
</tr>
</tbody>
</table>

Configuration

Please use `ttgo-t-watch` ID for `board` option in "`platformio.ini` (Project Configuration File):

```ini
[env:ttgo-t-watch]
platform = espressif32
board = ttgo-t-watch
```

You can override default TTGO T-Watch settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ttgo-t-watch.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ttgo-t-watch]
platform = espressif32
board = ttgo-t-watch

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

TTGO T-Watch supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:ttgo-t-watch]
platform = espressif32
board = ttgo-t-watch

upload_protocol = esptool
```
Debugging

Debugging currently does not support TTGO T-Watch board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

TTGO T1

Contents

- TTGO T1
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
<td>TTGO</td>
</tr>
</tbody>
</table>

Configuration

Please use ttgo-t1 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ttgo-t1]
platform = espressif32
board = ttgo-t1
```
You can override default TTGO T1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ttgo-t1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ttgo-t1]
platform = espressif32
board = ttgo-t1

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### Uploading

TTGO T1 supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `iot-bus-jtag`
- `jlink`
- `minimodule`
- `olimex-arm-usb-ocd`
- `olimex-arm-usb-ocd-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
- `tumpa`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:ttgo-t1]
platform = espressif32
board = ttgo-t1

upload_protocol = esptool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
TTGO T1 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

TinyPICO

Contents

- TinyPICO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TinyPICO</td>
</tr>
</tbody>
</table>
Configuration

Please use tinypico ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:tinypico]
platform = espressif32
board = tinypico
```

You can override default TinyPICO settings per build environment using board_*** option, where *** is a JSON object path from board manifest tinypico.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:tinypico]
platform = espressif32
board = tinypico

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

TinyPICO supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```ini
[env:tinypico]
platform = espressif32
board = tinypico

upload_protocol = esptool
```

Debugging

Debugging currently does not support TinyPICO board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
Turta IoT Node

Contents

• Turta IoT Node
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Turta</td>
</tr>
</tbody>
</table>

Configuration

Please use turta_iot_node ID for board option in “platformio.ini” (Project Configuration File):

```
[env:turta_iot_node]
platform = espressif32
board = turta_iot_node
```

You can override default Turta IoT Node settings per build environment using board_*** option, where *** is a JSON object path from board manifest turta_iot_node.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:turta_iot_node]
platform = espressif32
board = turta_iot_node

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Turta IoT Node supports the next uploading protocols:
• espota
• esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```plaintext
[env:turta_iot_node]
platform = espressif32
board = turta_iot_node

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support Turta IoT Node board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**VintLabs ESP32 Devkit**

**Contents**

- VintLabs ESP32 Devkit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
### Configuration

Please use `vintlabs-devkit-v1` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:vintlabs-devkit-v1]
platform = espressif32
board = vintlabs-devkit-v1
```

You can override default VintLabs ESP32 Devkit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `vintlabs-devkit-v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:vintlabs-devkit-v1]
platform = espressif32
board = vintlabs-devkit-v1

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### Uploading

VintLabs ESP32 Devkit supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:
[env:vintlabs-devkit-v1]
platform = espressif32
board = vintlabs-devkit-v1
upload_protocol = esptool

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

VintLabs ESP32 Devkit does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espresif IoT</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**WEMOS LOLIN D32**

**Contents**

- WEMOS LOLIN D32
  - Hardware
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use `lolink_d32` ID for `board` option in “`platformio.ini` (Project Configuration File):`

```
[env:lolink_d32]
platform = espressif32
board = lolink_d32
```

You can override default WEMOS LOLIN D32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lolink_d32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lolink_d32]
platform = espressif32
board = lolink_d32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

WEMOS LOLIN D32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32
upload_protocol = esptool
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

WEMOS LOLIN D32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

WEMOS LOLIN D32 PRO

Contents

- WEMOS LOLIN D32 PRO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use `lolin_d32_pro` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```ini
[env:lolin_d32_pro]
platform = espressif32
board = lolin_d32_pro
```

You can override default WEMOS LOLIN D32 PRO settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lolin_d32_pro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Upgrading

WEMOS LOLIN D32 PRO supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
WEMOS LOLIN D32 PRO does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESPRESSIF IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**WEMOS LOLIN32**

**Contents**

- **WEMOS LOLIN32**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>
Configuration

Please use lolin32 ID for board option in “platformio.ini” (Project Configuration File):

```Ini
[env:lolin32]
platform = espressif32
board = lolin32
```

You can override default WEMOS LOLIN32 settings per build environment using board_*** option, where *** is a JSON object path from board manifest lolin32.json. For example, board_build.mcu, board_build.f_cpu, etc.

```Ini
[env:lolin32]
platform = espressif32
board = lolin32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

WEMOS LOLIN32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```Ini
[env:lolin32]
platform = espressif32
board = lolin32

upload_protocol = esptool
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

⚠️ Warning: ⚠️ You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

WEMOS LOLIN32 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

WeMos D1 MINI ESP32

Contents

- WeMos D1 MINI ESP32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use `wemos_d1_mini32` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:wemos_d1_mini32]
platform = espressif32
board = wemos_d1_mini32
```

You can override default WeMos D1 MINI ESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wemos_d1_mini32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wemos_d1_mini32]
platform = espressif32
board = wemos_d1_mini32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

WeMos D1 MINI ESP32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:wemos_d1_mini32]
platform = espressif32
board = wemos_d1_mini32
upload_protocol = esptool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” *(Project Configuration File)*.

WeMos D1 MINI ESP32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**WeMos WiFi and Bluetooth Battery**
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use `wemosbat` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:wemosbat]
platform = espressif32
board = wemosbat
```

You can override default WeMos WiFi and Bluetooth Battery settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wemosbat.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wemosbat]
platform = espressif32
board = wemosbat

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

WeMos WiFi and Bluetooth Battery supports the next uploading protocols:

- esp-prog
- espota
• `esptool`
• `iot-bus-jtag`
• `jlink`
• `minimodule`
• `olimex-arm-usb-oct`
• `olimex-arm-usb-oct-h`
• `olimex-arm-usb-tiny-h`
• `olimex-jtag-tiny`
• `tumpa`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:wemosbat]
platform = espressif32
board = wemosbat
upload_protocol = esptool
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.

WeMos WiFi and Bluetooth Battery does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

Widora AIR

Contents

- Widora AIR
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Widora</td>
</tr>
</tbody>
</table>

Configuration

Please use widora-air ID for board option in “platformio.ini” (Project Configuration File):

```
[env:widora-air]
platform = espressif32
board = widora-air
```

You can override default Widora AIR settings per build environment using board_*** option, where *** is a JSON object path from board manifest widora-air.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

Widora AIR supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
platform = espressif32
board = widora-air
upload_protocol = esptool
```

Debugging

Debugging currently does not support Widora AIR board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

XinaBox CW02

Contents

- XinaBox CW02
  - Hardware
Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>XinaBox</td>
</tr>
</tbody>
</table>

Configuration

Please use `xinabox_cw02` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:xinabox_cw02]
platform = espressif32
board = xinabox_cw02
```

You can override default XinaBox CW02 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `xinabox_cw02.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xinabox_cw02]
platform = espressif32
board = xinabox_cw02

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

XinaBox CW02 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:xinabox_cw02]
platform = espressif32
board = xinabox_cw02
upload_protocol = esptool
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

XinaBox CW02 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

YeaCreate NSCREEN-32

Contents

- YeaCreate NSCREEN-32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>YeaCreate</td>
</tr>
</tbody>
</table>

Configuration

Please use `nscreen-32` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nscreen-32]
platform = espressif32
board = nscreen-32
```

You can override default YeaCreate NSCREEN-32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nscreen-32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

YeaCreate NSCREEN-32 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```yaml
[env:nscreen-32]
platform = espressif32
board = nscreen-32

upload_protocol = esptool
```

Debugging

*Debugging* currently does not support YeaCreate NSCREEN-32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

oddWires IoT-Bus Io

Contents

- oddWires IoT-Bus Io
  - Hardware
Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>oddWires</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `iotbusio` ID for `board` option in `platformio.ini` (Project Configuration File):

```
[env:iotbusio]
platform = espressif32
board = iotbusio
```

You can override default `oddWires IoT-Bus Io` settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `iotbusio.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:iotbusio]
platform = espressif32
board = iotbusio

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

**Uploading**

`oddWires IoT-Bus Io` supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `iot-bus-jtag`
- `jlink`
• minimodule
• olimex-arm-usb-ocd
• olimex-arm-usb-ocd-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:iotbusio]
platform = espressif32
board = iotbusio
upload_protocol = esptool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.

oddWires IoT-Bus Io does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

oddWires IoT-Bus Proteus

Contents

- oddWires IoT-Bus Proteus
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 32: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>oddWires</td>
</tr>
</tbody>
</table>

Configuration

Please use `iotbusproteus` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:iotbusproteus]
platform = espressif32
board = iotbusproteus
```

You can override default oddWires IoT-Bus Proteus settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `iotbusproteus.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
oddWires IoT-Bus Proteus supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-oct
- olimex-arm-usb-oct-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:iotbusproteus]
platform = espressif32
board = iotbusproteus
upload_protocol = esptool
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).
oddWires IoT-Bus Proteus does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP-Prog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>

**u-blox NINA-W10 series**

### Contents

- **u-blox NINA-W10 series**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Espressif 32**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>240MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>u-blox</td>
</tr>
</tbody>
</table>
Configuration

Please use `nina_w10` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:nina_w10]
platform = espressif32
board = nina_w10
```

You can override default u-blox NINA-W10 series settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nina_w10.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nina_w10]
platform = espressif32
board = nina_w10

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

u-blox NINA-W10 series supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:nina_w10]
platform = espressif32
board = nina_w10

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support u-blox NINA-W10 series board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
</tbody>
</table>
1.12.8 Espressif 8266

4D Systems gen4 IoD Range

Contents

- 4D Systems gen4 IoD Range
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>4D Systems</td>
</tr>
</tbody>
</table>

Configuration

Please use gen4iod ID for board option in "platformio.ini" (Project Configuration File):

```
[env:gen4iod]
platform = espressif8266
board = gen4iod
```

You can override default 4D Systems gen4 IoD Range settings per build environment using board_*** option, where *** is a JSON object path from board manifest gen4iod.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:gen4iod]
platform = espressif8266
board = gen4iod

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```
Uploading

4D Systems gen4 IoD Range supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:gen4iod]
platform = espressif8266
board = gen4iod
upload_protocol = esptool
```

Debugging

Debugging currently does not support 4D Systems gen4 IoD Range board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Adafruit HUZZAH ESP8266

Contents

- Adafruit HUZZAH ESP8266
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use huzzah ID for *board* option in “`platformio.ini` (Project Configuration File)”: `huzzah`

```ini
[env:huzzah]
platform = espressif8266
board = huzzah

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

You can override default Adafruit HUZZAH ESP8266 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `huzzah.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

Uploading

Adafruit HUZZAH ESP8266 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:huzzah]
platform = espressif8266
board = huzzah

upload_protocol = esptool
```
Debugging

Debugging currently does not support Adafruit HUZZAH ESP8266 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266 Non-OS SDK</strong></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266 RTOS SDK</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
<tr>
<td><strong>Simba</strong></td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

DigiStump Oak

Contents

- DigiStump Oak
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>DigiStump</td>
</tr>
</tbody>
</table>

Configuration

Please use oak ID for board option in “platformio.ini” (Project Configuration File):
You can override default DigiStump Oak settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `oak.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
platform = espressif8266
board = oak

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Uploading

DigiStump Oak supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```plaintext
platform = espressif8266
board = oak

upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support DigiStump Oak board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### ESP-Mx DevKit (ESP8285)
• **ESP-Mx DevKit (ESP8285)**
  
  **Hardware**

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Doit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `espxmxdevkit` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:espxmxdevkit]
platform = espressif8266
board = espxmxdevkit
```

You can override default ESP-Mx DevKit (ESP8285) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `espmxdevkit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:espxmxdevkit]
platform = espressif8266
board = espxmxdevkit

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ESP-Mx DevKit (ESP8285) supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```py
[env:espmxdevkit]
platform = espressif8266
board = espmxdevkit

upload_protocol = esptool
```

**Debugging**

`Debugging` currently does not support ESP-Mx DevKit (ESP8285) board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**ESP-WROOM-02**

**Contents**

- `ESP-WROOM-02`
  - **Hardware**
  - **Configuration**
  - **Uploading**
  - **Debugging**
  - **Frameworks**

**Hardware**

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `esp_wroom_02` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:esp_wroom_02]
platform = espressif8266
board = esp_wroom_02
```

You can override default ESP-WROOM-02 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `esp_wroom_02.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp_wroom_02]
platform = espressif8266
board = esp_wroom_02

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ESP-WROOM-02 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:esp_wroom_02]
platform = espressif8266
board = esp_wroom_02

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support ESP-WROOM-02 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

ESPDuino (ESP-13 Module)

Contents

- ESPDuino (ESP-13 Module)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Doit</td>
</tr>
</tbody>
</table>

Configuration

Please use espduino ID for *board* option in “platformio.ini” (Project Configuration File):

```
[env:espduino]
platform = espressif8266
board = espduino
```
You can override default ESPDuino (ESP-13 Module) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `espduino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:espduino]
platform = espressif8266
board = espduino

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ESPDuino (ESP-13 Module) supports the next uploading protocols:

- `esptota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:espduino]
platform = espressif8266
board = espduino

upload_protocol = esptool
```

**Debugging**

`Debugging` currently does not support ESPDuino (ESP-13 Module) board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**ESPectro Core**
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Dycodex</td>
</tr>
</tbody>
</table>

Configuration

Please use *espectro* ID for *board* option in “platformio.ini” (*Project Configuration File)*:

```
[env:espectro]
platform = espessif8266
board = espectro
```

You can override default ESPectro Core settings per build environment using *board_**** option, where *** is a JSON object path from board manifest espectro.json. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:espectro]
platform = espessif8266
board = espectro

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ESPectro Core supports the next uploading protocols:

- *espota*
- *esptool*
Default protocol is `esptool`.

You can change upload protocol using `upload_protocol` option:

```yaml
[env:espectro]
platform = espressif8266
board = espectro
upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support ESPectro Core board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached</td>
</tr>
<tr>
<td></td>
<td>to a wide range of Arduino boards to create all kinds of creative coding, interactive objects,</td>
</tr>
<tr>
<td></td>
<td>spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266</strong></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 func-</td>
</tr>
<tr>
<td><strong>Non-OS</strong></td>
<td>tionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface</td>
</tr>
<tr>
<td><strong>SDK</strong></td>
<td>functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
<tr>
<td><strong>RTOS</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SDK</strong></td>
<td></td>
</tr>
</tbody>
</table>

### ESPino

#### Contents

- ESPino
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ESPino</td>
</tr>
</tbody>
</table>

## Configuration

Please use espino ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:espino]
platform = espressif8266
board = espino
```

You can override default ESPino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `espino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:espino]
platform = espressif8266
board = espino

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

## Uploading

ESPino supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:espino]
platform = espressif8266
board = espino

upload_protocol = esptool
```

## Debugging

Debugging currently does not support ESPino board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266 Non-OS SDK</strong></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266 RTOS SDK</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

ESPRESSO Lite 1.0

Contents

- ESPRESSO Lite 1.0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Espressif 8266**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ESPert</td>
</tr>
</tbody>
</table>

Configuration

Please use `espresso_lite_v1` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:espresso_lite_v1]
platform = espressif8266
board = espresso_lite_v1
```
You can override default ESPresso Lite 1.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `espresso_lite_v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:espresso_lite_v1]
platform = espressif8266
board = espresso_lite_v1

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ESPRESSO Lite 1.0 supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:espresso_lite_v1]
platform = espressif8266
board = espresso_lite_v1

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support ESPRESSO Lite 1.0 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266 Non-OS SDK</strong></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266 RTOS SDK</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**ESPRESSO Lite 2.0**
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ESPert</td>
</tr>
</tbody>
</table>

Configuration

Please use `espresso_lite_v2` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:espresso_lite_v2]
platform = espressif8266
board = espresso_lite_v2
```

You can override default ESPresso Lite 2.0 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `espresso_lite_v2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:espresso_lite_v2]
platform = espressif8266
board = espresso_lite_v2

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ESPRESSo Lite 2.0 supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:espresso_lite_v2]
platform = espressif8266
board = espresso_lite_v2
upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support ESPresso Lite 2.0 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**Espressif ESP8266 ESP-12E**

**Contents**

- *Espressif ESP8266 ESP-12E*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `esp12e` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:esp12e]
platform = espressif8266
board = esp12e
```

You can override default Espressif ESP8266 ESP-12E settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp12e.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:esp12e]
platform = espressif8266
board = esp12e

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

Espressif ESP8266 ESP-12E supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:esp12e]
platform = espressif8266
board = esp12e

upload_protocol = esptool
```

**Debugging**

`Debugging` currently does not support Espressif ESP8266 ESP-12E board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Arduino</em></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><em>ESP8266</em></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><em>Simba</em></td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

Espressif Generic ESP8266 ESP-01 1M

Contents

- _Espressif Generic ESP8266 ESP-01 1M_
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform _Espressif 8266_: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp01_1m` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:esp01_1m]
platform = espressif8266
board = esp01_1m
```
You can override default Espressif Generic ESP8266 ESP-01 1M settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp01_1m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp01_1m]
platform = espressif8266
board = esp01_1m

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

Espressif Generic ESP8266 ESP-01 1M supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:esp01_1m]
platform = espressif8266
board = esp01_1m

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support Espressif Generic ESP8266 ESP-01 1M board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266 Non-OS SDK</strong></td>
<td>The non-Os SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266 RTOS SDK</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**Espressif Generic ESP8266 ESP-01 512k**
Contents

• Espressif Generic ESP8266 ESP-01 512k
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use esp01 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:esp01]
platform = espressif8266
board = esp01
```

You can override default Espressif Generic ESP8266 ESP-01 512k settings per build environment using board_*** option, where *** is a JSON object path from board manifest esp01.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:esp01]
platform = espressif8266
board = esp01

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Espressif Generic ESP8266 ESP-01 512k supports the next uploading protocols:

• espota
• esptool
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp01]
platform  = espressif8266
board    = esp01
upload_protocol  = esptool
```

### Debugging

`Debugging` currently does not support Espressif Generic ESP8266 ESP-01 512k board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

### Espressif Generic ESP8266 ESP-07 1MB

#### Contents

- Espressif Generic ESP8266 ESP-07 1MB
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `esp07` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:esp07]
platform = espressif8266
board = esp07
```

You can override default Espressif Generic ESP8266 ESP-07 1MB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp07.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp07]
platform = espressif8266
board = esp07

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

Espressif Generic ESP8266 ESP-07 1MB supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp07]
platform = espressif8266
board = esp07

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support Espressif Generic ESP8266 ESP-07 1MB board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Espressif Generic ESP8266 ESP-07S

Contents

- Espressif Generic ESP8266 ESP-07S
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp07s` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:esp07s]
platform = espressif8266
board = esp07s
```
You can override default Espressif Generic ESP8266 ESP-07S settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp07s.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp07s]
platform = espressif8266
board = esp07s

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

Espressif Generic ESP8266 ESP-07S supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp07s]
platform = espressif8266
board = esp07s

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support Espressif Generic ESP8266 ESP-07S board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**Generic ESP8285 Module**
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp8285` ID for *board* option in "platformio.ini" (*Project Configuration File*):

```
[env:esp8285]
platform = espressif8266
board = esp8285
```

You can override default Generic ESP8285 Module settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp8285.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp8285]
platform = espressif8266
board = esp8285

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Generic ESP8285 Module supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is `esptool`.

You can change upload protocol using `upload_protocol` option:

```python
[env:esp8285]
platform = espressif8266
board = esp8285
upload_protocol = esptool
```

### Debugging

Debugging currently does not support Generic ESP825 Module board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266 Non-OS SDK</strong></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266 RTOS SDK</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

### Heltec Wifi kit 8

**Contents**

- Heltec Wifi kit 8
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Espresif 8266**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
**Configuration**

Please use `heltec_wifi_kit_8` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:heltec_wifi_kit_8]
platform = espressif8266
board = heltec_wifi_kit_8
```

You can override default Heltec Wifi kit 8 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `heltec_wifi_kit_8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:heltec_wifi_kit_8]
platform = espressif8266
board = heltec_wifi_kit_8

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

Heltec Wifi kit 8 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:heltec_wifi_kit_8]
platform = espressif8266
board = heltec_wifi_kit_8

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support Heltec Wifi kit 8 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Invent One

Contents

- Invent One
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Invent One</td>
</tr>
</tbody>
</table>

Configuration

Please use `inventone` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:inventone]
platform = espressif8266
board = inventone
```

You can override default Invent One settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `inventone.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.
PlatformIO Documentation, Release 5.0.5a1

Uploading

Invent One supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```python
[env:inventone]
platform = espressif8266
board = inventone

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support Invent One board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

NodeMCU 0.9 (ESP-12 Module)
### Contents

- NodeMCU 0.9 (ESP-12 Module)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Espressif 8266**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NodeMCU</td>
</tr>
</tbody>
</table>

### Configuration

Please use `nodemcu` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:nodemcu]
platform = espressif8266
board = nodemcu
```

You can override default NodeMCU 0.9 (ESP-12 Module) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nodemcu.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nodemcu]
platform = espressif8266
board = nodemcu

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Uploading

NodeMCU 0.9 (ESP-12 Module) supports the next uploading protocols:

- espota
- esptool
Default protocol is **esptool**

You can change upload protocol using `upload_protocol` option:

```python
[env:nodemcu]
platform = espressif8266
board = nodemcu
upload_protocol = esptool
```

## Debugging

*Debugging* currently does not support NodeMCU 0.9 (ESP-12 Module) board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

### NodeMCU 1.0 (ESP-12E Module)

#### Contents

- NodeMCU 1.0 (ESP-12E Module)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
## PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NodeMCU</td>
</tr>
</tbody>
</table>

## Configuration

Please use `nodemcuv2` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nodemcuv2]
platform = espressif8266
board = nodemcuv2
```

You can override default NodeMCU 1.0 (ESP-12E Module) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nodemcuv2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nodemcuv2]
platform = espressif8266
board = nodemcuv2

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

## Uploading

NodeMCU 1.0 (ESP-12E Module) supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:nodemcuv2]
platform = espressif8266
board = nodemcuv2

upload_protocol = esptool
```

## Debugging

*Debugging* currently does not support NodeMCU 1.0 (ESP-12E Module) board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

Olimex MOD-WIFI-ESP8266(-DEV)

Contents

- Olimex MOD-WIFI-ESP8266(-DEV)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Olimex</td>
</tr>
</tbody>
</table>

Configuration

Please use modwifi ID for board option in “platformio.ini” (Project Configuration File):

```
[env:modwifi]
platform = espressif8266
board = modwifi
```
You can override default Olimex MOD-WIFI-ESP8266(-DEV) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `modwifi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:modwifi]
platform = espressif8266
board = modwifi

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Uploading

Olimex MOD-WIFI-ESP8266(-DEV) supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:modwifi]
platform = espressif8266
board = modwifi

upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support Olimex MOD-WIFI-ESP8266(-DEV) board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

### Phoenix 1.0
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use `phoenix_v1` ID for `board` option in “`platformio.ini`” (*Project Configuration File)*:

```ini
[env:phoenix_v1]
platform = espressif8266
board = phoenix_v1
```

You can override default Phoenix 1.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `phoenix_v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:phoenix_v1]
platform = espressif8266
board = phoenix_v1

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Phoenix 1.0 supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:phoenix_v1]
platform = espressif8266
board = phoenix_v1
upload_protocol = esptool
```

**Debugging**

`Debugging` currently does not support Phoenix 1.0 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**Phoenix 2.0**

**Contents**

- Phoenix 2.0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Espessif 8266**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

### Configuration

Please use `phoenix_v2` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:phoenix_v2]
platform = espressif8266
board = phoenix_v2
```

You can override default Phoenix 2.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `phoenix_v2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:phoenix_v2]
platform = espressif8266
board = phoenix_v2

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000
```

### Uploading

Phoenix 2.0 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:phoenix_v2]
platform = espressif8266
board = phoenix_v2

upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support Phoenix 2.0 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Schirmilabs Eduino WiFi

Contents

- Schirmilabs Eduino WiFi
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espassif 8266: Espassif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Schirmilabs</td>
</tr>
</tbody>
</table>

Configuration

Please use eduinowifi ID for board option in “platformio.ini” (Project Configuration File):

```
[env:eduinowifi]
platform = espressif8266
board = eduinowifi
```

1.12. Boards
You can override default Schirmilabs Eduino WiFi settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `eduinowifi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```json
[env:eduinowifi]
  platform = espressif8266
  board = eduinowifi

  ; change microcontroller
  board_build.mcu = esp8266

  ; change MCU frequency
  board_build.f_cpu = 80000000L
```

**Uploading**

Schirmilabs Eduino WiFi supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```json
[env:eduinowifi]
  platform = espressif8266
  board = eduinowifi

  upload_protocol = esptool
```

**Debugging**

`Debugging` currently does not support Schirmilabs Eduino WiFi board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**Sonoff Basic**
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ITEAD</td>
</tr>
</tbody>
</table>

Configuration

Please use `sonoff_basic` ID for `board` option in “platformio.ini” (*Project Configuration File*):

```ini
[env:sonoff_basic]
platform = espressif8266
board = sonoff_basic
```

You can override default Sonoff Basic settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sonoff_basic.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sonoff_basic]
platform = espressif8266
board = sonoff_basic

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Sonoff Basic supports the next uploading protocols:

- espota
- esptool
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:sonoff_basic]
platform = espressif8266
board = sonoff_basic
upload_protocol = esptool
```

**Debugging**

`Debugging` currently does not support Sonoff Basic board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266 Non-OS SDK</strong></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266 RTOS SDK</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**Sonoff S20**

**Contents**

- **Sonoff S20**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform `Espressif 8266`: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
## Configuration

Please use `sonoff_s20` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:sonoff_s20]
platform = espressif8266
board = sonoff_s20
```

You can override default Sonoff S20 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sonoff_s20.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sonoff_s20]
platform = espressif8266
board = sonoff_s20

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

## Uploading

Sonoff S20 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:sonoff_s20]
platform = espressif8266
board = sonoff_s20

upload_protocol = esptool
```

## Debugging

*Debugging* currently does not support Sonoff S20 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Sonoff SV

Contents

- Sonoff SV
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ITEAD</td>
</tr>
</tbody>
</table>

Configuration

Please use sonoff_sv ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sonoff_sv]
platform = espressif8266
board = sonoff_sv
```

You can override default Sonoff SV settings per build environment using board_*** option, where *** is a JSON object path from board manifest sonoff_sv.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:sonoff_sv]
platform = espressif8266
board = sonoff_sv

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L

Uploading

Sonoff SV supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

[env:sonoff_sv]
platform = espressif8266
board = sonoff_sv

upload_protocol = esptool

Debugging

Debugging currently does not support Sonoff SV board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Sonoff TH

Contents
• Sonoff TH
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ITEAD</td>
</tr>
</tbody>
</table>

Configuration

Please use `sonoff_th` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:sonoff_th]
platform = espressif8266
board = sonoff_th
```

You can override default Sonoff TH settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sonoff_th.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sonoff_th]
platform = espressif8266
board = sonoff_th

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Sonoff TH supports the next uploading protocols:

• espota
• esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:
[env:sonoff_th]
platform = espressif8266
board = sonoff_th
upload_protocol = esptool

Debugging

*Debugging* currently does not support Sonoff TH board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>ESP8266 Non-OS SDK</strong></td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td><strong>ESP8266 RTOS SDK</strong></td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

SparkFun Blynk Board

Contents

- SparkFun Blynk Board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>
Configuration

Please use sparkfunBlynk ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:sparkfunBlynk]
platform = espressif8266
board = sparkfunBlynk
```

You can override default SparkFun Blynk Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfunBlynk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparkfunBlynk]
platform = espressif8266
board = sparkfunBlynk

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

SparkFun Blynk Board supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:sparkfunBlynk]
platform = espressif8266
board = sparkfunBlynk

upload_protocol = esptool
```

Debugging

`Debugging` currently does not support SparkFun Blynk Board board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

SparkFun ESP8266 Thing

Contents

- SparkFun ESP8266 Thing
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espresif 8266: Espresif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use thing ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:thing]
platform = espresif8266
board = thing
```
You can override default SparkFun ESP8266 Thing settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `thing.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:thing]
platform = espressif8266
board = thing

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Uploading**

SparkFun ESP8266 Thing supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:thing]
platform = espressif8266
board = thing

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support SparkFun ESP8266 Thing board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**SparkFun ESP8266 Thing Dev**
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use `thingdev` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:thingdev]
platform = espressif8266
board = thingdev
```

You can override default SparkFun ESP8266 Thing Dev settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `thingdev.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:thingdev]
platform = espressif8266
board = thingdev

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

SparkFun ESP8266 Thing Dev supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is \texttt{esptool}

You can change upload protocol using \texttt{upload_protocol} option:

```
[env:thingdev]
platform = espressif8266
board = thingdev
upload_protocol = esptool
```

**Debugging**

\textit{Debugging} currently does not support SparkFun ESP8266 Thing Dev board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

**SweetPea ESP-210**

**Contents**

- \textit{SweetPea ESP-210}
  - Hardware
  - \textit{Configuration}
  - Uploading
  - \textit{Debugging}
  - Frameworks

**Hardware**

Platform \textit{Espressif 8266}: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SweetPea</td>
</tr>
</tbody>
</table>

### Configuration

Please use `esp210` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:esp210]
platform = espressif8266
board = esp210
```

You can override default SweetPea ESP-210 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp210.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:esp210]
platform = espressif8266
board = esp210

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Uploading

SweetPea ESP-210 supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:esp210]
platform = espressif8266
board = esp210

upload_protocol = esptool
```

### Debugging

*Debugging* currently does not support SweetPea ESP-210 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

ThaiEasyElec ESPino

Contents

- ThaiEasyElec ESPino
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ThaiEasyElec</td>
</tr>
</tbody>
</table>

Configuration

Please use `espinotee` ID for board option in “platformio.ini” (Project Configuration File):

```
[env:espinotee]
platform = espressif8266
board = espinotee
```
You can override default ThaiEasyElec ESPino settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `espinotee.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```json
[env:espinotee]
platform = espressif8266
board = espinotee

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Uploading

ThaiEasyElec ESPino supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```json
[env:espinotee]
platform = espressif8266
board = espinotee

upload_protocol = esptool
```

### Debugging

*Debugging* currently does not support ThaiEasyElec ESPino board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

### WEMOS D1 R1

### 1.12. Boards
Hardware

Platform **Espressif 8266**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use `d1` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:d1]
platform = espressif8266
board = d1
```

You can override default WEMOS D1 R1 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `d1.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:d1]
platform = espressif8266
board = d1

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

WEMOS D1 R1 supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:d1]
platform = espressif8266
board = d1
upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support WEMOS D1 R1 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

### WeMos D1 R2 and mini

#### Contents

- WeMos D1 R2 and mini
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform `Espressif 8266`: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
# Configuration

Please use `d1_mini` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:d1_mini]
platform = espressif8266
board = d1_mini
```

You can override default WeMos D1 R2 and mini settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `d1_mini.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:d1_mini]
platform = espressif8266
board = d1_mini

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

# Uploading

WeMos D1 R2 and mini supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:d1_mini]
platform = espressif8266
board = d1_mini

upload_protocol = esptool
```

# Debugging

*Debugging* currently does not support WeMos D1 R2 and mini board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

WeMos D1 mini Lite

Contents

- WeMos D1 mini Lite
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use `d1_mini_lite` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:d1_mini_lite]
platform = espressif8266
board = d1_mini_lite
```
You can override default WeMos D1 mini Lite settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `d1_mini_lite.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:d1_mini_lite]
platform = espressif8266
board = d1_mini_lite

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

WeMos D1 mini Lite supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:d1_mini_lite]
platform = espressif8266
board = d1_mini_lite

upload_protocol = esptool
```

**Debugging**

*Debugging* currently does not support WeMos D1 mini Lite board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td><a href="https://www.arduino.cc/en/Reference/UI">Arduino Wiring-based Framework</a> allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**WeMos D1 mini Pro**

**Contents**

- WeMos D1 mini Pro
  - Hardware
Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use `d1_mini_pro` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:d1_mini_pro]
platform = espressif8266
board = d1_mini_pro
```

You can override default WeMos D1 mini Pro settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `d1_mini_pro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:d1_mini_pro]
platform = espressif8266
board = d1_mini_pro

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

WeMos D1 mini Pro supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool.

You can change upload protocol using `upload_protocol` option:
DEBUGGING

Debugging currently does not support WeMos D1 mini Pro board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

WiFi Slot

Contents

- WiFi Slot
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Amperka</td>
</tr>
</tbody>
</table>
Configuration

Please use wifi_slot ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:wifi_slot]
platform = espressif8266
board = wifi_slot
```

You can override default WiFi Slot settings per build environment using board_*** option, where *** is a JSON object path from board manifest wifi_slot.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:wifi_slot]
platform = espressif8266
board = wifi_slot

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

WiFi Slot supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```ini
[env:wifi_slot]
platform = espressif8266
board = wifi_slot

upload_protocol = esptool
```

Debugging

Debugging currently does not support WiFi Slot board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

WiFiduino

Contents

- WiFiduino
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WiFiduino</td>
</tr>
</tbody>
</table>

Configuration

Please use wifi duino ID for board option in “platformio.ini” (Project Configuration File):

```
[env:wifi duino]
platform = espressif8266
board = wifi duino
```

You can override default WiFiduino settings per build environment using board_*** option, where *** is a JSON object path from board manifest wifi duino.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

WiFiduino supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:wifiduino]
platform = espressif8266
board = wifiduino
upload_protocol = esptool
```

Debugging

*Debugging* currently does not support WiFiduino board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

WifInfo
PlatformIO Documentation, Release 5.0.5a1

- **WifInfo**
  - **Hardware**
  - **Configuration**
  - **Uploading**
  - **Debugging**
  - **Frameworks**

### Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espressif</td>
</tr>
</tbody>
</table>

### Configuration

Please use *wifinfo* ID for *board* option in *"platformio.ini" (Project Configuration File)*:

```
[env:wifinfo]
platform = espressif8266
board = wifinfo
```

You can override default WifInfo settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *wifinfo.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:wifinfo]
platform = espressif8266
board = wifinfo

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Uploading

WifInfo supports the next uploading protocols:

- espota
- esptool

Default protocol is *esptool*

You can change upload protocol using *upload_protocol* option:
Debugging

Debugging currently does not support WifInfo board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Wio Link

Contents

- Wio Link
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Espressif 8266: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>
Configuration

Please use `wio_link` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:wio_link]
platform = espressif8266
board = wio_link
```

You can override default Wio Link settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wio_link.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:wio_link]
platform = espressif8266
board = wio_link
board_build.mcu = esp8266
board_build.f_cpu = 80000000L
```

Uploading

Wio Link supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:wio_link]
platform = espressif8266
board = wio_link
upload_protocol = esptool
```

Debugging

`Debugging` currently does not support Wio Link board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

Wio Node

Contents

- **Wio Node**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Espressif 8266**: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>

Configuration

Please use `wio_node` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:wio_node]
platform = espressif8266
board = wio_node
```

You can override default Wio Node settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wio_node.json`. For example, `board_build.mcu,board_build.f_cpu, etc.`
Uploading

Wio Node supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:wio_node]
platform = espressif8266
board = wio_node
upload_protocol = esptool
```

Debugging

Debugging currently does not support Wio Node board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

XinaBox CW01

Contents
• XinaBox CW01
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform *Espressif 8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>XinaBox</td>
</tr>
</tbody>
</table>

Configuration

Please use xinabox_cw01 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:xinabox_cw01]
platform = espressif8266
board = xinabox_cw01
```

You can override default XinaBox CW01 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest xinabox_cw01.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:xinabox_cw01]
platform = espressif8266
board = xinabox_cw01

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

XinaBox CW01 supports the next uploading protocols:

• espota
• esptool
Default protocol is `esptool`.

You can change upload protocol using `upload_protocol` option:

```
[env:xinabox_cw01]
platform = espressif8266
board = xinabox_cw01
upload_protocol = esptool
```

### Debugging

`Debugging` currently does not support XinaBox CW01 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP8266 Non-OS SDK</td>
<td>The non-OS SDK provides a set of application programming interfaces (APIs) for core ESP8266 functionalities such as data reception/transmission over Wi-Fi, TCP/IP stack functions, hardware interface functions and basic system management functions</td>
</tr>
<tr>
<td>ESP8266 RTOS SDK</td>
<td>ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers</td>
</tr>
</tbody>
</table>

### 1.12.9 Freescale Kinetis

#### Ethernet IoT Starter Kit

**Contents**

- Ethernet IoT Starter Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Freescale Kinetis*: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.
Microcontroller | MK64FN1M0VLL12
--- | ---
Frequency | 120MHz
Flash | 1MB
RAM | 256KB
Vendor | Freescale

**Configuration**

Please use IBMEthernetKit ID for **board** option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:IBMEthernetKit]
platform = freescalekinetis
board = IBMEthernetKit
```

You can override default Ethernet IoT Starter Kit settings per build environment using **board_*** option, where *** is a JSON object path from board manifest IBMEthernetKit.json. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```ini
[env:IBMEthernetKit]
platform = freescalekinetis
board = IBMEthernetKit

; change microcontroller
board_build.mcu = mk64fn1m0vll12

; change MCU frequency
board_build.f_cpu = 12000000L
```

**Uploading**

Ethernet IoT Starter Kit supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using **upload_protocol** option:

```ini
[env:IBMEthernetKit]
platform = freescalekinetis
board = IBMEthernetKit

upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Ethernet IoT Starter Kit has on-board debug probe and is ready for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-K20D50M

Contents

- Freescale Kinetis FRDM-K20D50M
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK20DX128VLH5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>
Configuration

Please use frdm_k20d50m ID for board option in “platformio.ini” (Project Configuration File):

```
[env:frdm_k20d50m]
platform = freescalekinetis
board = frdm_k20d50m
```

You can override default Freescale Kinetis FRDM-K20D50M settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_k20d50m.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:frdm_k20d50m]
platform = freescalekinetis
board = frdm_k20d50m

; change microcontroller
board_build.mcu = mk20dx128vlh5

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Freescale Kinetis FRDM-K20D50M supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:frdm_k20d50m]
platform = freescalekinetis
board = frdm_k20d50m

upload_protocol = mbed
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-K20D50M has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Freescale Kinetis FRDM-K22F

#### Contents

- Freescale Kinetis FRDM-K22F
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK22FN512VLH12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

#### Configuration

Please use frdm_k22f ID for board option in “platformio.ini” (Project Configuration File):

```python
[env:frdm_k22f]
platform = freescalekinetis
board = frdm_k22f
```
You can override default Freescale Kinetis FRDM-K22F settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `frdm_k22f.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:frdm_k22f]
platform = freescalekinetis
board = frdm_k22f

; change microcontroller
board_build.mcu = mk22fn512v1h12

; change MCU frequency
board_build.f_cpu = 120000000L
```

### Uploading

Freescale Kinetis FRDM-K22F supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:frdm_k22f]
platform = freescalekinetis
board = frdm_k22f

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-K22F has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-K64F

Contents

- Freescale Kinetis FRDM-K64F
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK64FN1M0VLL12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use `frdm_k64f` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:frdm_k64f]
platform = freescalekinetis
board = frdm_k64f
```

You can override default Freescale Kinetis FRDM-K64F settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `frdm_k64f.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Freescale Kinetis FRDM-K64F supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:frdm_k64f]
platform = freescalekinetis
board = frdm_k64f

upload_protocol = mbed
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-K64F has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-K66F

Contents

- Freescale Kinetis FRDM-K66F
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK66FN2M0VMD18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_k66f ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:frdm_k66f]
platform = freescalekinetis
board = frdm_k66f
```

You can override default Freescale Kinetis FRDM-K66F settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_k66f.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:frdm_k66f]
platform = freescalekinetis
board = frdm_k66f

; change microcontroller
board_build.mcu = mk66fn2m0vmd18

; change MCU frequency
board_build.f_cpu = 180000000L

Uploading

Freescale Kinetis FRDM-K66F supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

[env:frdm_k66f]
platform = freescalekinetis
board = frdm_k66f

upload_protocol = mbed

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” *(Project Configuration File)*.

Freescale Kinetis FRDM-K66F has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-K82F

Contents

- Freescale Kinetis FRDM-K82F
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK82FN256VLL15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>150MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_k82f ID for board option in “platformio.ini” (Project Configuration File):

```
[env:frdm_k82f]
platform = freescalekinetis
board = frdm_k82f
```

You can override default Freescale Kinetis FRDM-K82F settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_k82f.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:frdm_k82f]
platform = freescalekinetis
board = frdm_k82f
```

(continues on next page)
### Uploading

Freescale Kinetis FRDM-K82F supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:frdm_k82f]
platform = freescalekinetis
board = frdm_k82f
upload_protocol = mbed
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Freescale Kinetis FRDM-K82F has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-KL05Z

Contents

- Freescale Kinetis FRDM-KL05Z
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKL05Z32VFM4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_kl05z ID for board option in “platformio.ini” (Project Configuration File):

```
[env:frdm_kl05z]
platform = freescalekinetis
board = frdm_kl05z
```

You can override default Freescale Kinetis FRDM-KL05Z settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_kl05z.json. For example, board_build.mcu, board_build.f_cpu, etc.
### Uploading

Freescale Kinetis FRDM-KL05Z supports the next uploading protocols:

- `cmsis-dap`
- `jlink`
- `mbed`

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:frdm_kl05z]
platform = freescalekinetis
board = frdm_kl05z

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-KL05Z has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-KL25Z

Contents

- Freescale Kinetis FRDM-KL25Z
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKL25Z128VLK4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_kl25z ID for board option in “platformio.ini” (Project Configuration File):

```plaintext
[env:frdm_kl25z]
platform = freescalekinetis
board = frdm_kl25z
```

You can override default Freescale Kinetis FRDM-KL25Z settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_kl25z.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:frdm_kl25z]
platform = freescalekinetis
board = frdm_kl25z
```

(continues on next page)
Uploading

Freescale Kinetis FRDM-KL25Z supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```ini
[env:frdm_kl25z]
platform = freescalekinetis
board = frdm_kl25z
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (*Project Configuration File*).

Freescale Kinetis FRDM-KL25Z has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-KL27Z

Contents

- Freescale Kinetis FRDM-KL27Z
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKL27Z64VLH4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_kl27z ID for board option in “platformio.ini” (Project Configuration File):

```
[env:frdm_kl27z]
platform = freescalekinetis
board = frdm_kl27z
```

You can override default Freescale Kinetis FRDM-KL27Z settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_kl27z.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:frdm_kl27z]
platform = freescalekinetis
board = frdm_kl27z

; change microcontroller
board_build.mcu = mkl27z64vlh4

; change MCU frequency
board_build.f_cpu = 48000000L

Uploading

Freescale Kinetis FRDM-KL27Z supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

[env:frdm_kl27z]
platform = freescalekinetis
board = frdm_kl27z

upload_protocol = mbed

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-KL27Z has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-KL43Z

Contents

- Freescale Kinetis FRDM-KL43Z
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKL43Z256VLH4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_kl43z ID for board option in “platformio.ini” (Project Configuration File):

```
[env:frdm_kl43z]
platform = freescalekinetis
board = frdm_kl43z
```

You can override default Freescale Kinetis FRDM-KL43Z settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_kl43z.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:frdm_kl43z]
platform = freescalekinetis
board = frdm_kl43z
```
; change microcontroller
board_build.mcu = mk143z256vlh4

; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

Freescale Kinetis FRDM-KL43Z supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is **mbed**

You can change upload protocol using `upload_protocol` option:

```ini
[env:frdm_kl43z]
platform = freescalekinetis
board = frdm_kl43z
upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Freescale Kinetis FRDM-KL43Z has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>
Freescale Kinetis FRDM-KL46Z

Contents

- Freescale Kinetis FRDM-KL46Z
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKL46Z256VLL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_kl46z ID for board option in “platformio.ini” (Project Configuration File):

```
[env:frdm_kl46z]
platform = freescalekinetis
board = frdm_kl46z
```

You can override default Freescale Kinetis FRDM-KL46Z settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_kl46z.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:frdm_kl46z]
platform = freescalekinetis
board = frdm_kl46z

; change microcontroller
board_build.mcu = mkl46z256vll4

; change MCU frequency
board_build.f_cpu = 48000000L
```
Uploading

Freescale Kinetis FRDM-KL46Z supports the next uploading protocols:
  • cmsis-dap
  • jlink
  • mbed
Default protocol is mbed
You can change upload protocol using upload_protocol option:

```ini
[env:frdm_kl46z]
platform = freescalekinetis
board = frdm_kl46z
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-KL46Z has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Mbed</em></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-KL82Z

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Freescale Kinetis FRDM-KL82Z</em></td>
</tr>
</tbody>
</table>
Hardware

Platform *Freescale Kinetis*: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKL82Z128VLK7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use `frdm_kl82z` ID for *board* option in “`platformio.ini` (Project Configuration File)“:

```ini
[env:frdm_kl82z]
platform = freescalekinetis
board = frdm_kl82z
```

You can override default Freescale Kinetis FRDM-KL82Z settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `frdm_kl82z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:frdm_kl82z]
platform = freescalekinetis
board = frdm_kl82z

; change microcontroller
board_build.mcu = mkl82z128vlk7

; change MCU frequency
board_build.f_cpu = 96000000L
```

Uploading

Freescale Kinetis FRDM-KL82Z supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed
You can change upload protocol using *upload_protocol* option:

```ini
[env:frdm_kl82z]
platform = freescalekinetis
board = frdm_kl82z
upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Freescale Kinetis FRDM-KL82Z does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**Freescale Kinetis FRDM-KW24D512**

**Contents**

- *Freescale Kinetis FRDM-KW24D512*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKW24D512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Freescale</td>
</tr>
</tbody>
</table>

Configuration

Please use frdm_kw24d ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:frdm_kw24d]
platform = freescalekinetis
board = frdm_kw24d
```

You can override default Freescale Kinetis FRDM-KW24D512 settings per build environment using board_*** option, where *** is a JSON object path from board manifest frdm_kw24d.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:frdm_kw24d]
platform = freescalekinetis
board = frdm_kw24d

; change microcontroller
board_build.mcu = mkw24d512

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Freescale Kinetis FRDM-KW24D512 supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```ini
[env:frdm_kw24d]
platform = freescalekinetis
board = frdm_kw24d

upload_protocol = mbed
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-KW24D512 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephy</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-KW41Z

Contents

- Freescale Kinetis FRDM-KW41Z
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.
### Configuration

Please use `frdm_kw41z` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:frdm_kw41z]
platform = freescalekinetis
board = frdm_kw41z
```

You can override default Freescale Kinetis FRDM-KW41Z settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `frdm_kw41z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:frdm_kw41z]
platform = freescalekinetis
board = frdm_kw41z

; change microcontroller
board_build.mcu = mkw41z512vht4

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Freescale Kinetis FRDM-KW41Z supports the next uploading protocols:

- `cmsis-dap`
- `jlink`
- `mbed`

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:frdm_kw41z]
platform = freescalekinetis
board = frdm_kw41z

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Freescale Kinetis FRDM-KW41Z has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
</tbody>
</table>

Hexiwear

Contents

- Hexiwear
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Freescale Kinetis: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK64FN1M0VDC12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MikroElektronika</td>
</tr>
</tbody>
</table>

1.12. Boards
Configuration

Please use `hexiwear` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:hexiwear]
platform = freescalekinetis
board = hexiwear
```

You can override default Hexiwear settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hexiwear.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:hexiwear]
platform = freescalekinetis
board = hexiwear

; change microcontroller
board_build.mcu = mk64fn1m0vdc12

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Hexiwear supports the next uploading protocols:
- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:hexiwear]
platform = freescalekinetis
board = hexiwear

upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Hexiwear does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephy RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### SEGGER IP Switch Board

#### Contents

- SEGGER IP Switch Board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **Freescale Kinetis**: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK66FN2M0VMD18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SEGGER</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `segger_ip_switch` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:segger_ip_switch]
platform = freescalekinetis
board = segger_ip_switch
```
You can override default SEGGER IP Switch Board settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `segger_ip_switch.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:segger_ip_switch]
platform = freescalekinetis
board = segger_ip_switch

; change microcontroller
board_build.mcu = mk66fn2m0vmd18

; change MCU frequency
board_build.f_cpu = 180000000L
```

### Uploading

SEGGER IP Switch Board supports the next uploading protocols:

- `jlink`
- `mbed`

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:segger_ip_switch]
platform = freescalekinetis
board = segger_ip_switch

upload_protocol = jlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

SEGGER IP Switch Board does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

1.12.10 GigaDevice GD32V

GD32VF103V-EVAL

Contents

- GD32VF103V-EVAL
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **GigaDevice GD32V**: The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GD32VF103VBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>108MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>

Configuration

Please use `gd32vf103v-eval` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:gd32vf103v-eval]
platform = gd32v
board = gd32vf103v-eval
```

You can override default GD32VF103V-EVAL settings per build environment using `board_``` option, where ``` is a JSON object path from board manifest `gd32vf103v-eval.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:gd32vf103v-eval]
platform = gd32v
board = gd32vf103v-eval
```

(continues on next page)
; change microcontroller
board_build.mcu = GD32VF103VBT6

; change MCU frequency
board_build.f_cpu = 108000000L

Uploading

GD32VF103V-EVAL supports the next uploading protocols:

- altera-usb-blaster
- gd-link
- jlink
- rv-link
- serial
- sipeed-rv-debugger
- um232h

Default protocol is gd-link

You can change upload protocol using upload_protocol option:

```
[env:gd32vf103v-eval]
platform = gd32v
board = gd32vf103v-eval
upload_protocol = gd-link
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using debug_tool option in “platformio.ini” (Project Configuration File).

GD32VF103V-EVAL does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

- **Altera / Intel USB-Blaster**
- **GD-LINK**
- **J-LINK**
- **RV-LINK**
- **Sipeed RV Debugger**
- **UM232H**

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>GigaDevice GD32V SDK</strong></td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
</tbody>
</table>

### Sipeed Longan Nano

#### Contents

- **Sipeed Longan Nano**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **GigaDevice GD32V**: The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GD32VF103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>108MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>
Configuration

Please use `sipeed-longan-nano` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:sipeed-longan-nano]
platform = gd32v
board = sipeed-longan-nano
```

You can override default Sipeed Longan Nano settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sipeed-longan-nano.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sipeed-longan-nano]
platform = gd32v
board = sipeed-longan-nano

; change microcontroller
board_build.mcu = GD32VF103CBT6

; change MCU frequency
board_build.f_cpu = 108000000L
```

Uploading

Sipeed Longan Nano supports the next uploading protocols:

- `altera-usb-blaster`
- `gd-link`
- `jlink`
- `rv-link`
- `serial`
- `sipeed-rv-debugger`
- `um232h`

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:sipeed-longan-nano]
platform = gd32v
board = sipeed-longan-nano

upload_protocol = serial
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sipeed Longan Nano does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera / Intel USB-Blaster</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>GD-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RV-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UM232H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice GD32V SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
</tbody>
</table>

### Sipeed Longan Nano Lite

**Contents**

- **Sipeed Longan Nano Lite**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **GigaDevice GD32V**: The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GD32VF103C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>108MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sipeed-longan-nano-lite` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:sipeed-longan-nano-lite]
platform = gd32v
board = sipeed-longan-nano-lite
```

You can override default Sipeed Longan Nano Lite settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sipeed-longan-nano-lite.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sipeed-longan-nano-lite]
platform = gd32v
board = sipeed-longan-nano-lite

; change microcontroller
board_build.mcu = GD32VF103C8T6

; change MCU frequency
board_build.f_cpu = 108000000L
```

**Uploading**

Sipeed Longan Nano Lite supports the next uploading protocols:

- `altera-usb-blaster`
- `gd-link`
- `jlink`
- `rv-link`
- `serial`
- `sipeed-rv-debugger`
- `um232h`

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:sipeed-longan-nano-lite]
platform = gd32v
board = sipeed-longan-nano-lite

upload_protocol = serial
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sipeed Longan Nano Lite does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera / Intel USB-Blaster</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>GD-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RV-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UM232H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice GD32V SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
</tbody>
</table>

Wio Lite RISC-V

- **Wio Lite RISC-V**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **GigaDevice GD32V**: The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GD32VF103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>108MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>

Configuration

Please use `wio_lite_risc-v` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:wio_lite_risc-v]
platform = gd32v
board = wio_lite_risc-v
```

You can override default Wio Lite RISC-V settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wio_lite_risc-v.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wio_lite_risc-v]
platform = gd32v
board = wio_lite_risc-v

; change microcontroller
board_build.mcu = GD32VF103CBT6

; change MCU frequency
board_build.f_cpu = 108000000L
```

Uploading

Wio Lite RISC-V supports the next uploading protocols:

- `altera-usb-blaster`
- `gd-link`
- `jlink`
- `rv-link`
- `serial`
- `sipeed-rv-debugger`
- `um232h`

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
[env:wio_lite_risc-v]
platform = gd32v
board = wio_lite_risc-v
upload_protocol = serial

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Wio Lite RISC-V does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera / Intel USB-Blaster</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>GD-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RV-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UM232H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice GD32V SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
</tbody>
</table>

**1.12.11 Infineon XMC**

**XMC1100 Boot Kit**

**Contents**

- **XMC1100 Boot Kit**
Hardware

Platform *Infineon XMC*: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>XMC1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

Configuration

Please use `xmc1100_boot_kit` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:xmc1100_boot_kit]
platform = infineonxmc
board = xmc1100_boot_kit
```

You can override default XMC1100 Boot Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1100_boot_kit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc1100_boot_kit]
platform = infineonxmc
board = xmc1100_boot_kit

; change microcontroller
board_build.mcu = XMC1100

; change MCU frequency
board_build.f_cpu = 32000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).
XMC1100 Boot Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**XMC1100 H-Bridge 2Go**

**Contents**

- **XMC1100 H-Bridge 2Go**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

**Hardware**

Platform **Infineon XMC**: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>XMC1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `xmc1100_h_bridge2go` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:xmc1100_h_bridge2go]
platform = infineonxmc
board = xmc1100_h_bridge2go
```

You can override default XMC1100 H-Bridge 2Go settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1100_h_bridge2go.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:xmc1100_h_bridge2go]
platform = infineonxmc
board = xmc1100_h_bridge2go

; change microcontroller
board_build.mcu = XMC1100

; change MCU frequency
board_build.f_cpu = 32000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

| Warning: | You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information. |

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File)*.

XMC1100 H-Bridge 2Go has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

XMC1100 XMC2Go

Contents

- *XMC1100 XMC2Go*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform *Infineon XMC*: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>XMC1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

Configuration

Please use `xmc1100_xmc2go` ID for `board` option in “`platformio.ini` (Project Configuration File):`

```
[env:xmc1100_xmc2go]
platform = infineonxmc
board = xmc1100_xmc2go
```

You can override default XMC1100 XMC2Go settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1100_xmc2go.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc1100_xmc2go]
platform = infineonxmc
board = xmc1100_xmc2go

; change microcontroller
board_build.mcu = XMC1100

; change MCU frequency
board_build.f_cpu = 32000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

XMC1100 XMC2Go has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>J-LINK</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

XMC1300 Boot Kit

Contents

- **XMC1300 Boot Kit**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

Hardware

Platform **Infineon XMC**: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>XMC1300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

Configuration

Please use `xmc1300_boot_kit` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:xmc1300_boot_kit]
platform = infineonxmc
board = xmc1300_boot_kit
```

You can override default XMC1300 Boot Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `xmc1300_boot_kit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:xmc1300_boot_kit]
platform = infineonxmc
board = xmc1300_boot_kit

; change microcontroller
board_build.mcu = XMC1300
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

XMC1300 Boot Kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

XMC1300 Sense2GoL

Hardware

Platform Infineon XMC: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform
Microcontroller | XMC1300  
---|---  
Frequency | 32MHz  
Flash | 32KB  
RAM | 16KB  
Vendor | Infineon

### Configuration

Please use `xmc1300_sense2gol` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:xmc1300_sense2gol]
platform = infineonxmc
board = xmc1300_sense2gol
```

You can override default XMC1300 Sense2GoL settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1300_sense2gol.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:xmc1300_sense2gol]
platform = infineonxmc
board = xmc1300_sense2gol

; change microcontroller
board_build.mcu = XMC1300

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

XMC1300 Sense2GoL has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

XMC1400 Boot Kit

Contents

- XMC1400 Boot Kit
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Infineon XMC**: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>XMC1400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.95MB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

Configuration

Please use `xmc1400_boot_kit` ID for `board` option in “`platformio.ini` (Project Configuration File):”

```
[env:xmc1400_boot_kit]
platform = infineonxmc
board = xmc1400_boot_kit
```

You can override default XMC1400 Boot Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1400_boot_kit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc1400_boot_kit]
platform = infineonxmc
board = xmc1400_boot_kit

; change microcontroller
board_build.mcu = XMC1400
```

(continues on next page)
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

XMC1400 Boot Kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

XMC4200 Distance2Go

Contents

- **XMC4200 Distance2Go**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Infineon XMC**: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.
### Configuration

Please use `xmc4200_distance2go` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:xmc4200_distance2go]
platform = infineonxmc
board = xmc4200_distance2go
```

You can override default XMC4200 Distance2Go settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `xmc4200_distance2go.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc4200_distance2go]
platform = infineonxmc
board = xmc4200_distance2go

; change microcontroller
board_build.mcu = XMC4200

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

XMC4200 Distance2Go has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

1.12. Boards

---

PlatformIO Documentation, Release 5.0.5a1

---

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>XMC4200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>40KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

---
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

XMC4700 Relax Kit

Contents

- **XMC4700 Relax Kit**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Infineon XMC*: Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>XMC4700</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>144MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2.00MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1.95MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

Configuration

Please use `xmc4700_relax_kit` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:xmc4700_relax_kit]
platform = infineonxmc
board = xmc4700_relax_kit
```

You can override default XMC4700 Relax Kit settings per build environment using `board.*` option, where `*` is a JSON object path from board manifest `xmc4700_relax_kit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:xmc4700_relax_kit]
platform = infineonxmc
board = xmc4700_relax_kit

; change microcontroller
board_build.mcu = XMC4700
```

(continues on next page)
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

XMC4700 Relax Kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

1.12.12 Intel ARC32

Arduino/Genuino 101

Contents

- *Arduino/Genuino 101*
  - *Hardware*
  - *Configuration*
  - *Debugging*
  - *Frameworks*
Hardware

Platform *Intel ARC32*: ARC embedded processors are a family of 32-bit CPUs that are widely used in SoC devices for storage, home, mobile, automotive, and Internet of Things applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ARCV2EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>152KB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Intel</td>
</tr>
</tbody>
</table>

Configuration

Please use *genuino101* ID for *board* option in *“platformio.ini” (Project Configuration File)*:

```Ini
[env:genuino101]
platform = intel_arc32
board = genuino101
```

You can override default Arduino/Genuino 101 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *genuino101.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```Ini
[env:genuino101]
platform = intel_arc32
board = genuino101

; change microcontroller
board_build.mcu = ARCV2EM

; change MCU frequency
board_build.f_cpu = 32000000L
```

Debugging

*Debugging* currently does not support Arduino/Genuino 101 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

1.12.13 Intel MCS-51 (8051)

Generic N79E8432
Hardware

Platform *Intel MCS-51 (8051)*: The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>N79E8432</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>22MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nuvoton</td>
</tr>
</tbody>
</table>

Configuration

Please use `n79e8432` ID for *board* option in “`platformio.ini`” (*Project Configuration File*):

```
[env:n79e8432]
platform = intel_mcs51
board = n79e8432
```

You can override default Generic N79E8432 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `n79e8432.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:n79e8432]
platform = intel_mcs51
board = n79e8432

; change microcontroller
board_build.mcu = n79e8432

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

*Debugging* currently does not support Generic N79E8432 board.

Generic N79E844
Hardware

Platform **Intel MCS-51 (8051)**: The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>N79E844</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>22MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nuvoton</td>
</tr>
</tbody>
</table>

Configuration

Please use n79e844 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:n79e844]
platform = intel_mcs51
board = n79e844
```

You can override default Generic N79E844 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest n79e844.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:n79e844]
platform = intel_mcs51
board = n79e844

; change microcontroller
board_build.mcu = n79e844

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

*Debugging* currently does not support Generic N79E844 board.

Generic N79E845
Contents

- **Generic N79E845**
  - Hardware
  - Configuration
  - Debugging

Hardware

Platform *Intel MCS-51 (8051)*: The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>N79E845</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>22MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nuvoton</td>
</tr>
</tbody>
</table>

Configuration

Please use n79e845 ID for `board` option in “platformio.ini” (*Project Configuration File*):

```ini
[env:n79e845]
platform = intel_mcs51
board = n79e845
```

You can override default Generic N79E845 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest n79e845.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:n79e845]
platform = intel_mcs51
board = n79e845

; change microcontroller
board_build.mcu = n79e845

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

*Debugging* currently does not support Generic N79E845 board.

Generic N79E854
Hardware

Platform Intel MCS-51 (8051): The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>N79E854</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>22MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nuvoton</td>
</tr>
</tbody>
</table>

Configuration

Please use n79e854 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:n79e854]
platform = intel_mcs51
board = n79e854
```

You can override default Generic N79E854 settings per build environment using board_*** option, where *** is a JSON object path from board manifest n79e854.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:n79e854]
platform = intel_mcs51
board = n79e854

; change microcontroller
board_build.mcu = n79e854

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

Debugging currently does not support Generic N79E854 board.

Generic N79E855
Hardware

Platform *Intel MCS-51 (8051)*: The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>N79E855</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>22MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nuvoton</td>
</tr>
</tbody>
</table>

Configuration

Please use `n79e855` ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:n79e855]
platform = intel_mcs51
board = n79e855
```

You can override default Generic N79E855 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `n79e855.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:n79e855]
platform = intel_mcs51
board = n79e855

; change microcontroller
board_build.mcu = n79e855

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

*Debugging* currently does not support Generic N79E855 board.

Generic STC15F204EA
Contents

- Generic STC15F204EA
  - Hardware
  - Configuration
  - Debugging

Hardware

Platform Intel MCS-51 (8051): The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STC15F204EA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>11MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>STC</td>
</tr>
</tbody>
</table>

Configuration

Please use stc15f204ea ID for board option in “platformio.ini“ (Project Configuration File):

```
[env:stc15f204ea]
platform = intel_mcs51
board = stc15f204ea
```

You can override default Generic STC15F204EA settings per build environment using board_*** option, where *** is a JSON object path from board manifest stc15f204ea.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:stc15f204ea]
platform = intel_mcs51
board = stc15f204ea

; change microcontroller
board_build.mcu = stc15f204ea

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

Debugging currently does not support Generic STC15F204EA board.

Generic STC15F2K60S2
Hardware

Platform *Intel MCS-51 (8051)*: The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STC15F2K60S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>6MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>60KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>STC</td>
</tr>
</tbody>
</table>

Configuration

Please use `stc15f2k60s2` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:stc15f2k60s2]
platform = intel_mcs51
board = stc15f2k60s2
```

You can override default Generic STC15F2K60S2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest stc15f2k60s2.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stc15f2k60s2]
platform = intel_mcs51
board = stc15f2k60s2

; change microcontroller
board_build.mcu = stc15f2k60s2

; change MCU frequency
board_build.f_cpu = 6000000L
```

Debugging

*Debugging* currently does not support Generic STC15F2K60S2 board.

Generic STC15W204S
Hardware

Platform Intel MCS-51 (8051): The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (µC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STC15W204S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>11MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>STC</td>
</tr>
</tbody>
</table>

Configuration

Please use stc15w204s ID for board option in “platformio.ini” (Project Configuration File):

```
[env:stc15w204s]
platform = intel_mcs51
board = stc15w204s
```

You can override default Generic STC15W204S settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `stc15w204s.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stc15w204s]
platform = intel_mcs51
board = stc15w204s

; change microcontroller
board_build.mcu = stc15w204s

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

Debugging currently does not support Generic STC15W204S board.

Generic STC15W404AS
Hardware

Platform **Intel MCS-51 (8051)**: The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STC15W404AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>11MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>STC</td>
</tr>
</tbody>
</table>

Configuration

Please use stc15w404as ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:stc15w404as]
platform = intel_mcs51
board = stc15w404as
```

You can override default Generic STC15W404AS settings per build environment using `board_***` option, where *** is a JSON object path from board manifest stc15w404as.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stc15w404as]
platform = intel_mcs51
board = stc15w404as

; change microcontroller
board_build.mcu = stc15w404as

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

`Debugging` currently does not support Generic STC15W404AS board.

**Generic STC15W408AS**
Hardware

Platform Intel MCS-51 (8051): The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (μC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STC15W408AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>11MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>STC</td>
</tr>
</tbody>
</table>

Configuration

Please use stc15w408as ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:stc15w408as]
platform = intel_mcs51
board = stc15w408as
```

You can override default Generic STC15W408AS settings per build environment using board_*** option, where *** is a JSON object path from board manifest stc15w408as.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:stc15w408as]
platform = intel_mcs51
board = stc15w408as

; change microcontroller
board_build.mcu = stc15w408as

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

Debugging currently does not support Generic STC15W408AS board.

Generic STC89C52RC
Hardware

Platform Intel MCS-51 (8051): The Intel MCS-51 (commonly termed 8051) is an internally Harvard architecture, complex instruction set computer (CISC) instruction set, single chip microcontroller (uC) series developed by Intel in 1980 for use in embedded systems.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STC89C52RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>11MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>STC</td>
</tr>
</tbody>
</table>

Configuration

Please use stc89c52rc ID for board option in “platformio.ini” (Project Configuration File):

```
[env:stc89c52rc]
platform = intel_mcs51
board = stc89c52rc
```

You can override default Generic STC89C52RC settings per build environment using board_*** option, where *** is a JSON object path from board manifest stc89c52rc.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:stc89c52rc]
platform = intel_mcs51
board = stc89c52rc

; change microcontroller
board_build.mcu = stc89c52rc

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

Debugging currently does not support Generic STC89C52RC board.

1.12.14 Kendryte K210
Sipeed MAIX BiT

Contents

- Sipeed MAIX BiT
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Kendryte K210*: Kendryte K210 is an AI capable RISCV64 dual core SoC.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>K210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>6MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>

Configuration

Please use `sipeed-maix-bit` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:sipeed-maix-bit]
platform = kendryte210
board = sipeed-maix-bit
```

You can override default Sipeed MAIX BiT settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sipeed-maix-bit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sipeed-maix-bit]
platform = kendryte210
board = sipeed-maix-bit

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

Sipeed MAIX BiT supports the next uploading protocols:
• iot-bus-jtag
• jlink
• kflash
• minimodule
• olimex-arm-usb-oct
• olimex-arm-usb-oct-h
• olimex-arm-usb-tiny-h
• olimex-jtag-tiny
• sipeed-rv-debugger
• tumpa

Default protocol is **kflash**

You can change upload protocol using `upload_protocol` option:

```ini
[env:sipeed-maix-bit]
platform = kendryte210
board = sipeed-maix-bit
upload_protocol = kflash
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sipeed MAIX BiT does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>

Sipeed MAIX BiT with Mic

Contents

- Sipeed MAIX BiT with Mic
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Kendryte K210: Kendryte K210 is an AI capable RISCV64 dual core SoC.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>K210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>6MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>

Configuration

Please use sipeed-maix-bit-mic ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sipeed-maix-bit-mic]
platform = kendryte210
board = sipeed-maix-bit-mic
```

You can override default Sipeed MAIX BiT with Mic settings per build environment using board_*** option, where *** is a JSON object path from board manifest sipeed-maix-bit-mic.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

Sipeed MAIX BiT with Mic supports the next uploading protocols:

- iot-bus-jtag
- jlink
- kflash
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- sipeed-rv-debugger
- tumpa

Default protocol is kflash

You can change upload protocol using upload_protocol option:

```
[env:sipeed-maix-bit-mic]
platform = kendryte210
board = sipeed-maix-bit-mic

upload_protocol = kflash
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using debug_tool option in “platformio.ini” (Project Configuration File).

Sipeed MAIX BiT with Mic does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
## Compatible Tools

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>

## Sipeed MAIX GO

### Contents

- Sipeed MAIX GO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Kendryte K210**: Kendryte K210 is an AI capable RISCV64 dual core SoC.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>K210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>6MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>
Configuration

Please use `sipeed-maix-go` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:sipeed-maix-go]
platform = kendryte210
board = sipeed-maix-go
```

You can override default Sipeed MAIX GO settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sipeed-maix-go.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sipeed-maix-go]
platform = kendryte210
board = sipeed-maix-go

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

Sipeed MAIX GO supports the next uploading protocols:

- `iot-bus-jtag`
- `jlink`
- `kflash`
- `minimodule`
- `olimex-arm-usb-ocd`
- `olimex-arm-usb-ocd-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
- `sipeed-rv-debugger`
- `tumpa`

Default protocol is `kflash`

You can change upload protocol using `upload_protocol` option:

```
[env:sipeed-maix-go]
platform = kendryte210
board = sipeed-maix-go

upload_protocol = kflash
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

1.12. Boards
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Sipeed MAIX GO does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>

Sipeed MAIX ONE DOCK

Contents

- Sipeed MAIX ONE DOCK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **Kendryte K210**: Kendryte K210 is an AI capable RISCV64 dual core SoC.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>K210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>6MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>

Configuration

Please use `sipeed-maix-one-dock` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:sipeed-maix-one-dock]
platform = kendryte210
board = sipeed-maix-one-dock
```

You can override default Sipeed MAIX ONE DOCK settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sipeed-maix-one-dock.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:sipeed-maix-one-dock]
platform = kendryte210
board = sipeed-maix-one-dock

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

Sipeed MAIX ONE DOCK supports the next uploading protocols:

- `iot-bus-jtag`
- `jlink`
- `kflash`
- `minimodule`
- `olimex-arm-usb-ocd`
- `olimex-arm-usb-ocd-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
- `sipeed-rv-debugger`
- `tumpa`
Default protocol is `kflash`.

You can change upload protocol using `upload_protocol` option:

```
[env:sipeed-maix-one-dock]
platform = kendryte210
board = sipeed-maix-one-dock
upload_protocol = kflash
```

### Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sipeed MAIX ONE DOCK does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>
Sipeed MAIXDUINO

Contents

- Sipeed MAIXDUINO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Kendryte K210: Kendryte K210 is an AI capable RISCV64 dual core SoC.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>K210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>6MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>

Configuration

Please use sipeed-maixduino ID for board option in "platformio.ini (Project Configuration File):

```
[env:sipeed-maixduino]
platform = kendryte210
board = sipeed-maixduino
```

You can override default Sipeed MAIXDUINO settings per build environment using board_*** option, where *** is a JSON object path from board manifest sipeed-maixduino.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sipeed-maixduino]
platform = kendryte210
board = sipeed-maixduino

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

Sipeed MAIXDUINO supports the next uploading protocols:
PlatformIO Documentation, Release 5.0.5a1

- iot-bus-jtag
- jlink
- kflash
- minimodule
- olimex-arm-usb-oct
- olimex-arm-usb-oct-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- sipeed-rv-debugger
- tumpa

Default protocol is kflash

You can change upload protocol using `upload_protocol` option:

```
[env:sipeed-maixduino]
platform = kendryte210
board = sipeed-maixduino
upload_protocol = kflash
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sipeed MAIXDUINO does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>

## Sipeed MF1 MF1

### Contents

- Sipeed MF1 MF1
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Kendryte K210*: Kendryte K210 is an AI capable RISCV64 dual core SoC.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>K210</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>6MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sipeed</td>
</tr>
</tbody>
</table>

### Configuration

Please use sipeed-MF1 ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:sipeed-MF1]
platform = kendryte210
board = sipeed-MF1
```

You can override default Sipeed MF1 MF1 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sipeed-MF1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:sipeed-MF1]
platform = kendryte210
board = sipeed-MF1

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L

### Uploading

Sipeed MF1 MF1 supports the next uploading protocols:

- iot-bus-jtag
- jlink
- kflash
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- sipeed-rv-debugger
- tumpa

Default protocol is kflash

You can change upload protocol using `upload_protocol` option:

[env:sipeed-MF1]
platform = kendryte210
board = sipeed-MF1

upload_protocol = kflash

### Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sipeed MF1 MF1 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IOT-Bus JTAG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sipeed RV Debugger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>

1.12.15 Lattice iCE40

IceZUM Alhambra FPGA

Contents

- IceZUM Alhambra FPGA
  - Hardware
  - Configuration
  - Debugging

Hardware

Platform Lattice iCE40: The iCE40 family of ultra-low power, non-volatile FPGAs has five devices with densities ranging from 384 to 7680 Look-Up Tables (LUTs). In addition to LUT-based, low-cost programmable logic, these devices feature Embedded Block RAM (EBR), Non-volatile Configuration Memory (NVCM) and Phase Locked Loops (PLLs). These features allow the devices to be used in low-cost, high-volume consumer and system applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ICE40-HX1K-TQ144</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>12MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>FPGAwards</td>
</tr>
</tbody>
</table>

Configuration

Please use icezum ID for board option in “platformio.ini” (Project Configuration File):
You can override default IceZUM Alhambra FPGA settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `icezum.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:icezum]
platform = lattice_ice40
board = icezum

; change microcontroller
board_build.mcu = iCE40-HX1K-TQ144

; change MCU frequency
board_build.f_cpu = 12000000L
```

### Debugging

*Debugging* currently does not support IceZUM Alhambra FPGA board.

### Lattice iCEstick FPGA Evaluation Kit

#### Contents

- Lattice iCEstick FPGA Evaluation Kit
  - Hardware
  - Configuration
  - Debugging

#### Hardware

Platform *Lattice iCE40*: The iCE40 family of ultra-low power, non-volatile FPGAs has five devices with densities ranging from 384 to 7680 Look-Up Tables (LUTs). In addition to LUT-based, low-cost programmable logic, these devices feature Embedded Block RAM (EBR), Non-volatile Configuration Memory (NVCM) and Phase Locked Loops (PLLs). These features allow the devices to be used in low-cost, high-volume consumer and system applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ICE40-HX1K-TQ144</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>12MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Lattice</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `icestick` ID for `board` option in “`platformio.ini`” (Project Configuration File):
You can override default Lattice iCEstick FPGA Evaluation Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `icestick.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:icestick]
platform = lattice_ice40
board = icestick

; change microcontroller
board_build.mcu = iCE40-HX1K-TQ144

; change MCU frequency
board_build.f_cpu = 12000000L
```

### Debugging

`Debugging` currently does not support Lattice iCEstick FPGA Evaluation Kit board.

### 1.12.16 Linux ARM

#### Raspberry Pi 1 Model B

**Contents**

- Raspberry Pi 1 Model B
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **Linux ARM**: Linux ARM is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS X, Linux ARM) you can build native application for Linux ARM platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>BCM2835</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>700MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Raspberry Pi</td>
</tr>
</tbody>
</table>
Configuration

Please use raspberrypi_1b ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:raspberrypi_1b]
platform = linux_arm
board = raspberrypi_1b
```

You can override default Raspberry Pi 1 Model B settings per build environment using board_*** option, where *** is a JSON object path from board manifest raspberrypi_1b.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:raspberrypi_1b]
platform = linux_arm
board = raspberrypi_1b

; change microcontroller
board_build.mcu = bcm2835

; change MCU frequency
board_build.f_cpu = 700000000L
```

Debugging

Debugging currently does not support Raspberry Pi 1 Model B board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiringPi</td>
<td>WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry Pi. It’s designed to be familiar to people who have used the Arduino ‘wiring’ system</td>
</tr>
</tbody>
</table>

Raspberry Pi 2 Model B

Contents

- Raspberry Pi 2 Model B
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Linux ARM: Linux ARM is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS
X, Linux ARM) you can build native application for Linux ARM platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>BCM2836</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>900MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1GB</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Raspberry Pi</td>
</tr>
</tbody>
</table>

**Configuration**

Please use raspberrypi_2b ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:raspberrypi_2b]
platform = linux_arm
board = raspberrypi_2b
```

You can override default Raspberry Pi 2 Model B settings per build environment using board_*** option, where *** is a JSON object path from board manifest raspberrypi_2b.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:raspberrypi_2b]
platform = linux_arm
board = raspberrypi_2b

; change microcontroller
board_build.mcu = bcm2836

; change MCU frequency
board_build.f_cpu = 900000000L
```

**Debugging**

*Debugging* currently does not support Raspberry Pi 2 Model B board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WiringPi</strong></td>
<td>WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry Pi. It’s designed to be familiar to people who have used the Arduino ‘wiring’ system</td>
</tr>
</tbody>
</table>

**Raspberry Pi 3 Model B**

**Contents**

- Raspberry Pi 3 Model B
  - Hardware
  - Configuration
Hardware

Platform **Linux ARM**: Linux ARM is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS X, Linux ARM) you can build native application for Linux ARM platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>BCM2837</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1200MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1GB</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Raspberry Pi</td>
</tr>
</tbody>
</table>

Configuration

Please use `raspberrypi_3b` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:raspberrypi_3b]
platform = linux_arm
board = raspberrypi_3b
```

You can override default Raspberry Pi 3 Model B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `raspberrypi_3b.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:raspberrypi_3b]
platform = linux_arm
board = raspberrypi_3b

; change microcontroller
board_build.mcu = bcm2837

; change MCU frequency
board_build.f_cpu = 1200000000L
```

Debugging

*Debugging* currently does not support Raspberry Pi 3 Model B board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiringPi</td>
<td>WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry Pi. It’s designed to be familiar to people who have used the Arduino ‘wiring’ system</td>
</tr>
</tbody>
</table>
Raspberry Pi Zero

Contents

• Raspberry Pi Zero
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform **Linux ARM**: Linux ARM is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. Using host OS (Mac OS X, Linux ARM) you can build native application for Linux ARM platform.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>BCM2835</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1000MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Raspberry Pi</td>
</tr>
</tbody>
</table>

Configuration

Please use `raspberrypi_zero` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:raspberrypi_zero]
platform = linux_arm
board = raspberrypi_zero
```

You can override default Raspberry Pi Zero settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `raspberrypi_zero.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:raspberrypi_zero]
platform = linux_arm
board = raspberrypi_zero

; change microcontroller
board_build.mcu = bcm2835

; change MCU frequency
board_build.f_cpu = 1000000000L
```

Debugging

*Debugging* currently does not support Raspberry Pi Zero board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiringPi</td>
<td>WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry Pi. It’s designed to be familiar to people who have used the Arduino ‘wiring’ system</td>
</tr>
</tbody>
</table>

1.12.17 Maxim 32

MAX32620FTHR

Contents

- MAX32620FTHR
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Maxim 32: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32620FTHR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Maxim</td>
</tr>
</tbody>
</table>

Configuration

Please use `max32620f thr` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:max32620f thr]
platform = maxim32
board = max32620f thr
```

You can override default MAX32620FTHR settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `max32620f thr.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.
Uploading

MAX32620FTHR supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:max32620fthr
platform = maxim32
board = max32620fthr

upload_protocol = mbed
```

Debugging

_Debugging_ - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging _Tools & Debug Probes_ using debug_tool option in “platformio.ini” (Project Configuration File).

MAX32620FTHR does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Mbed</em></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>
MAX32625MBED

Contents

- MAX32625MBED
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Maxim 32*: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32625</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>160KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Maxim</td>
</tr>
</tbody>
</table>

Configuration

Please use `max32625mbed` ID for *board* option in “`platformio.ini`” (*Project Configuration File*):

```ini
[env:max32625mbed]
platform = maxim32
board = max32625mbed
```

You can override default MAX32625MBED settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `max32625mbed.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:max32625mbed]
platform = maxim32
board = max32625mbed

; change microcontroller
board_build.mcu = max32625

; change MCU frequency
board_build.f_cpu = 96000000L
```

Debugging

*Debugging* currently does not support MAX32625MBED board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

MAX32625NEXPAQ

Contents

- MAX32625NEXPAQ
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Maxim 32**: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32625</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>160KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Maxim</td>
</tr>
</tbody>
</table>

Configuration

Please use max32625nexpaq ID for board option in "platformio.ini" (Project Configuration File):

```
[env:max32625nexpaq]
platform = maxim32
board = max32625nexpaq
```

You can override default MAX32625NEXPAQ settings per build environment using board_*** option, where *** is a JSON object path from board manifest max32625nexpaq.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:max32625nexpaq]
platform = maxim32
board = max32625nexpaq
; change microcontroller
```
board_build.mcu = max32625
; change MCU frequency
board_build.f_cpu = 96000000L

Debugging

Debugging currently does not support MAX32625NEXPAQ board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

MAX32625PICO

Contents

- MAX32625PICO
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Maxim 32: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX32625</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>160KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Maxim</td>
</tr>
</tbody>
</table>

Configuration

Please use max32625pico ID for board option in “platformio.ini” (Project Configuration File):
You can override default MAX32625PICO settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `max32625pico.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:max32625pico]
platform = maxim32
board = max32625pico

; change microcontroller
board_build.mcu = max32625

; change MCU frequency
board_build.f_cpu = 96000000L
```

### Debugging

*Debugging* currently does not support MAX32625PICO board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Maxim ARM mbed Enabled Development Platform for MAX32600

#### Contents

- Maxim ARM mbed Enabled Development Platform for MAX32600
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform *Maxim 32*: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.
Microcontroller  |  MAX32600
---|---
Frequency  |  24MHz
Flash  |  256KB
RAM  |  32KB
Vendor  |  Maxim

### Configuration

Please use `max32600mbed` ID for `board` option in "`platformio.ini`" (Project Configuration File):

```ini
[env:max32600mbed]
platform = maxim32
board = max32600mbed
```

You can override default Maxim ARM mbed Enabled Development Platform for MAX32600 settings per build environment using `board_**` option, where `**` is a JSON object path from board manifest `max32600mbed.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:max32600mbed]
platform = maxim32
board = max32600mbed

; change microcontroller
board_build.mcu = max32600

; change MCU frequency
board_build.f_cpu = 24000000L
```

### Uploading

Maxim ARM mbed Enabled Development Platform for MAX32600 supports the next uploading protocols:

- `cmsis-dap`
- `mbed`

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:max32600mbed]
platform = maxim32
board = max32600mbed

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Maxim ARM mbed Enabled Development Platform for MAX32600 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Maxim Health Sensor Platform

#### Contents

- **Maxim Health Sensor Platform**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Maxim 32**: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32620</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Maxim</td>
</tr>
</tbody>
</table>

### Configuration

Please use `max32620hsp` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default Maxim Health Sensor Platform settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `max32620hsp.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:max32620hsp]
platform = maxim32
board = max32620hsp

; change microcontroller
board_build.mcu = max32620

; change MCU frequency
board_build.f_cpu = 96000000L
```

### Uploading

Maxim Health Sensor Platform supports the next uploading protocols:

- `jlink`
- `mbed`

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:max32620hsp]
platform = maxim32
board = max32620hsp

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini“ (Project Configuration File)*.

Maxim Health Sensor Platform does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Maxim MAX32630FTHR Application Platform

Contents

- Maxim MAX32630FTHR Application Platform
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Maxim 32: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32630</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Maxim</td>
</tr>
</tbody>
</table>

Configuration

Please use `max32630fthr` ID for `board` option in "`platformio.ini`" (Project Configuration File):

```ini
[env:max32630fthr]
platform = maxim32
board = max32630fthr
```

You can override default Maxim MAX32630FTHR Application Platform settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `max32630fthr.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:max32630fthr]
platform = maxim32
board = max32630fthr
; change microcontroller
```

(continues on next page)
board_build.mcu = `max32630`

; change MCU frequency
board_build.f_cpu = 96000000L

### Debugging

*Debugging* currently does not support Maxim MAX32630FTHR Application Platform board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Mbed</em></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Maxim Wireless Sensor Node Demonstrator

#### Contents

- *Maxim Wireless Sensor Node Demonstrator*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Maxim 32*: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32610</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>24MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Maxim</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `maxwsnenv` ID for `board` option in “platformio.ini” *(Project Configuration File)*:
You can override default Maxim Wireless Sensor Node Demonstrator settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `maxwsnenv.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:maxwsnenv]
platform = maxim32
board = maxwsnenv

; change microcontroller
board_build.mcu = max32610

; change MCU frequency
board_build.f_cpu = 24000000L
```

### Uploading

Maxim Wireless Sensor Node Demonstrator supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:maxwsnenv]
platform = maxim32
board = maxwsnenv

upload_protocol = mbed
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Maxim Wireless Sensor Node Demonstrator does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

1.12. Boards
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
</tbody>
</table>

SDT32620B

Contents

- SDT32620B
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Maxim 32**: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32620IWG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sigma Delta Technologies</td>
</tr>
</tbody>
</table>

Configuration

Please use `sdt32620b` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:sdt32620b]
platform = maxim32
board = sdt32620b
```

You can override default SDT32620B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sdt32620b.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:sdt32620b]
platform = maxim32
board = sdt32620b

; change microcontroller
board_build.mcu = max32620iwg
```

(continues on next page)
; change MCU frequency
board_build.f_cpu = 96000000L

Debugging

*Debugging* currently does not support SDT32620B board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

SDT32625B

Contents

- SDT32625B
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Maxim 32*: Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32625ITK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>160KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sigma Delta Technologies</td>
</tr>
</tbody>
</table>

Configuration

Please use `sdt32625b` ID for `board` option in “platformio.ini” *(Project Configuration File)*:
You can override default SDT32625B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sdt32625b.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:sdt32625b]
platform = maxim32
board = sdt32625b

; change microcontroller
board_build.mcu = max32625itk

; change MCU frequency
board_build.f_cpu = 96000000L
```

### Debugging

`Debugging` currently does not support SDT32625B board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### 1.12.18 Microchip PIC32

#### 4D Systems PICadillo 35T

**Contents**

- 4D Systems PICadillo 35T
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
### Configuration

Please use `picadillo_35t` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:picadillo_35t]
platform = microchippic32
board = picadillo_35t
```

You can override default 4D Systems PICadillo 35T settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `picadillo_35t.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:picadillo_35t]
platform = microchippic32
board = picadillo_35t

; change microcontroller
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

*Debugging* currently does not support 4D Systems PICadillo 35T board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### DataStation Mini

**Contents**

- *DataStation Mini*
  - Hardware
  - Configuration
Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX150F128C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>40MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Makerology</td>
</tr>
</tbody>
</table>

Configuration

Please use dsmini ID for board option in “platformio.ini” (Project Configuration File):

```
[env:dsmini]
platform = microchippic32
board = dsmini
```

You can override default DataStation Mini settings per build environment using board_*** option, where *** is a JSON object path from board manifest dsmini.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:dsmini]
platform = microchippic32
board = dsmini

; change microcontroller
board_build.mcu = 32MX150F128C

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

Debugging currently does not support DataStation Mini board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Digilent Cerebot 32MX4

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX460F512L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use `cerebot32mx4` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:cerebot32mx4]
platform = microchippic32
board = cerebot32mx4
```

You can override default Digilent Cerebot 32MX4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `cerebot32mx4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cerebot32mx4]
platform = microchippic32
board = cerebot32mx4

; change microcontroller
board_build.mcu = 32MX460F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

*Debugging* currently does not support Digilent Cerebot 32MX4 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digilent Cerebot 32MX7

Contents

- Digilent Cerebot 32MX7
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high-performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use cerebot32mx7 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:cerebot32mx7]
platform = microchippic32
board = cerebot32mx7
```

You can override default Digilent Cerebot 32MX7 settings per build environment using board_*** option, where *** is a JSON object path from board manifest cerebot32mx7.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:cerebot32mx7]
platform = microchippic32
board = cerebot32mx7
; change microcontroller
```

(continues on next page)
board_build.mcu = \texttt{32MX795F512L}  
\texttt{; change MCU frequency}
board_build.f_cpu = \texttt{80000000L}

\section*{Debugging}

\textit{Debugging} currently does not support Digilent Cerebot 32MX7 board.

\section*{Frameworks}

\begin{table}[h]
\centering
\begin{tabular}{|l|p{10cm}|}
\hline
Name & Description \\
\hline\hline
Arduino & Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences \\
\hline
\end{tabular}
\end{table}

\section*{Digilent OpenScope}

\subsection*{Contents}

\begin{itemize}
\item \textit{Digilent OpenScope}
  \begin{itemize}
  \item \textit{Hardware}
  \item \textit{Configuration}
  \item \textit{Debugging}
  \item \textit{Frameworks}
  \end{itemize}
\end{itemize}

\section*{Hardware}

Platform \textit{Microchip PIC32}: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

\begin{table}[h]
\centering
\begin{tabular}{|l|c|}
\hline
\textbf{Microcontroller} & \texttt{32MZ2048EFG124} \\
\textbf{Frequency} & 200MHz \\
\textbf{Flash} & 1.98MB \\
\textbf{RAM} & 512KB \\
\textbf{Vendor} & Digilent \\
\hline
\end{tabular}
\end{table}

\section*{Configuration}

Please use \texttt{opencode} ID for \texttt{board} option in “\texttt{platformio.ini}” (Project Configuration File):
[env:openscope]
platform = microchippic32
board = openscope

You can override default Digilent OpenScope settings per build environment using board_*** option, where *** is a JSON object path from board manifest openscope.json. For example, board_build.mcu, board_build.f_cpu, etc.

[env:openscope]
platform = microchippic32
board = openscope

; change microcontroller
board_build.mcu = 32MZ2048EFG124

; change MCU frequency
board_build.f_cpu = 200000000L

Debugging

Debugging currently does not support Digilent OpenScope board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digilent chipKIT Cmod

Contents

- Digilent chipKIT Cmod
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX150F128D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>40MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>124KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

**Configuration**

Please use chipkit_cmod ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:chipkit_cmod]
platform = microchippic32
board = chipkit_cmod
```

You can override default Digilent chipKIT Cmod settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chipkit_cmod.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:chipkit_cmod]
platform = microchippic32
board = chipkit_cmod

; change microcontroller
board_build.mcu = 32MX150F128D

; change MCU frequency
board_build.f_cpu = 40000000L
```

**Debugging**

*Debugging* currently does not support Digilent chipKIT Cmod board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Digilent chipKIT DP32**

**Contents**

- *Digilent chipKIT DP32*
  - Hardware
  - Configuration
Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX250F128B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>40MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use chipkit_dp32 ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:chipkit_dp32]
platform = microchippic32
board = chipkit_dp32
```

You can override default Digilent chipKIT DP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chipkit_dp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:chipkit_dp32]
platform = microchippic32
board = chipkit_dp32

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

*Debugging* currently does not support Digilent chipKIT DP32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Digilent chipKIT MAX32

Contents

- Digilent chipKIT MAX32
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX795F512L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use `mega_pic32` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:mega_pic32]
platform = microchippic32
board = mega_pic32
```

You can override default Digilent chipKIT MAX32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `mega_pic32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mega_pic32]
platform = microchippic32
board = mega_pic32

; change microcontroller
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

*Debugging* currently does not support Digilent chipKIT MAX32 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digilent chipKIT MX3

Contents

- Digilent chipKIT MX3
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX320F128H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>124KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use chipkit_mx3 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:chipkit_mx3]
platform = microchippic32
board = chipkit_mx3
```

You can override default Digilent chipKIT MX3 settings per build environment using board_*** option, where *** is a JSON object path from board manifest chipkit_mx3.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:chipkit_mx3]
platform = microchippic32
board = chipkit_mx3

; change microcontroller
```

(continues on next page)
board_build.mcu = 32MX320F128H

; change MCU frequency
board_build.f_cpu = 80000000L

Debugging

Debugging currently does not support Digilent chipKIT MX3 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digilent chipKIT Pro MX4

Contents

- Digilent chipKIT Pro MX4
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high-performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX460F512L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use chipkit_pro_mx4 ID for board option in “platformio.ini” (Project Configuration File):
You can override default Digilent chipKIT Pro MX4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `chipkit_pro_mx4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:chipkit_pro_mx4]
platform = microchippic32
board = chipkit_pro_mx4

; change microcontroller
board_build.mcu = 32MX460F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Debugging**

Debugging currently does not support Digilent chipKIT Pro MX4 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Digilent chipKIT Pro MX7**

**Contents**

- *Digilent chipKIT Pro MX7*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
**Microcontroller** 32MX795F512L
**Frequency** 80MHz
**Flash** 508KB
**RAM** 128KB
**Vendor** Digilent

**Configuration**

Please use chipkit_pro_mx7 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:chipkit_pro_mx7]
platform = microchippic32
board = chipkit_pro_mx7
```

You can override default Digilent chipKIT Pro MX7 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest chipkit_pro_mx7.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:chipkit_pro_mx7]
platform = microchippic32
board = chipkit_pro_mx7

; change microcontroller
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Debugging**

*Debugging* currently does not support Digilent chipKIT Pro MX7 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Digilent chipKIT UNO32**

**Contents**

- *Digilent chipKIT UNO32*
  - Hardware
  - Configuration
Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX320F128H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>124KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use `uno_pic32` ID for `board` option in “platformio.ini” (*Project Configuration File*):

```
[env:uno_pic32]
platform = microchippic32
board = uno_pic32
```

You can override default Digilent chipKIT UNO32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `uno_pic32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:uno_pic32]
platform = microchippic32
board = uno_pic32

; change microcontroller
board_build.mcu = 32MX320F128H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

*Debugging* currently does not support Digilent chipKIT UNO32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Digilent chipKIT WF32

Contents

- Digilent chipKIT WF32
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX695F512L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use chipkit_wf32 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:chipkit_wf32]
platform = microchippic32
board = chipkit_wf32
```

You can override default Digilent chipKIT WF32 settings per build environment using board_*** option, where *** is a JSON object path from board manifest chipkit_wf32.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:chipkit_wf32]
platform = microchippic32
board = chipkit_wf32

; change microcontroller
board_build.mcu = 32MX695F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

Debugging currently does not support Digilent chipKIT WF32 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
</tbody>
</table>

Digilent chipKIT WiFire

Contents

- Digilent chipKIT WiFire
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MZ2048ECG100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>200MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.98MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use **chipkit_wifire** ID for **board** option in "platformio.ini" (Project Configuration File):

```
[env:chipkit_wifire]
platform = microchippic32
board = chipkit_wifire
```

You can override default Digilent chipKIT WiFire settings per build environment using **board_*** option, where *** is a JSON object path from board manifest chipkit_wifire.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:chipkit_wifire]
platform = microchippic32
board = chipkit_wifire
; change microcontroller
```

(continues on next page)
PlatformIO Documentation, Release 5.0.5a1

(continued from previous page)

```plaintext
board_build.mcu = 32MZ2048ECG100
; change MCU frequency
board_build.f_cpu = 200000000L
```

Debugging

`Debugging` currently does not support Digilent chipKIT WiFire board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Digilent chipKIT uC32

Contents

- Digilent chipKIT uC32
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high-performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX340F512H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use `chipkit_uc32` ID for `board` option in “`platformio.ini`” (Project Configuration File):
You can override default Digilent chipKIT uC32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `chipkit_uc32.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```yaml
[env:chipkit_uc32]
platform = microchippic32
board = chipkit_uc32

; change microcontroller
board_build.mcu = 32MX340F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Debugging**

`Debugging` currently does not support Digilent chipKIT uC32 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Element14 chipKIT Pi**

**Contents**

- `Element14 chipKIT Pi`
  - `Hardware`
  - `Configuration`
  - `Debugging`
  - `Frameworks`

**Hardware**

Platform `Microchip PIC32`: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
## Configuration

Please use chipkit_pi ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:chipkit_pi]
platform = microchippic32
board = chipkit_pi
```

You can override default Element14 chipKIT Pi settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest chipkit_pi.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:chipkit_pi]
platform = microchippic32
board = chipkit_pi

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 40000000L
```

## Debugging

*Debugging* currently does not support Element14 chipKIT Pi board.

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## Fubarino Mini

### Contents

- Fubarino Mini
  - Hardware
  - Configuration
Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX250F128D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Fubarino</td>
</tr>
</tbody>
</table>

Configuration

Please use fubarino_mini ID for board option in “platformio.ini” (Project Configuration File):

```
[env:fubarino_mini]
platform = microchippic32
board = fubarino_mini
```

You can override default Fubarino Mini settings per build environment using board_*** option, where *** is a JSON object path from board manifest fubarino_mini.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:fubarino_mini]
platform = microchippic32
board = fubarino_mini

; change microcontroller
board_build.mcu = 32MX250F128D

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

Debugging currently does not support Fubarino Mini board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Fubarino SD (1.5)

Contents

- Fubarino SD (1.5)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high-performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX795F512H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Fubarino</td>
</tr>
</tbody>
</table>

Configuration

Please use fubarino_sd ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:fubarino_sd]
platform = microchippic32
board = fubarino_sd
```

You can override default Fubarino SD (1.5) settings per build environment using board_*** option, where *** is a JSON object path from board manifest fubarino_sd.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:fubarino_sd]
platform = microchippic32
board = fubarino_sd

; change microcontroller
board_build.mcu = 32MX795F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

Debugging currently does not support Fubarino SD (1.5) board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

HelvePic32

Contents

- HelvePic32
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX250F128B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BOXTEC</td>
</tr>
</tbody>
</table>

Configuration

Please use `helvepic32` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:helvepic32]
platform = microchippic32
board = helvepic32
```

You can override default HelvePic32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `helvepic32.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:helvepic32]
platform = microchippic32
board = helvepic32

; change microcontroller
board_build.mcu = 32MX250F128B
```

(continues on next page)


```plaintext
; change MCU frequency
board_build.f_cpu = 48000000L
```

## Debugging

*Debugging* currently does not support HelvePic32 board.

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### HelvePic32

#### Contents

- **HelvePic32**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX250F128B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BOXTEC</td>
</tr>
</tbody>
</table>

### Configuration

Please use `helvepic32_breadboardside ID` for `board` option in “platformio.ini” (*Project Configuration File)*:
You can override default HelvePic32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `helvepic32_breadboardside.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
board_build.mcu = 32MX250F128B
board_build.f_cpu = 48000000L
```

### Debugging

*Debugging* currently does not support HelvePic32 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### HelvePic32

**Contents**

- HelvePic32
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX250F128D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BOXTEC</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `helvepic32_smd` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:helvepic32_smd]
platform = microchippic32
board = helvepic32_smd
```

You can override default HelvePic32 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `helvepic32_smd.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.:  

```ini
[env:helvepic32_smd]
platform = microchippic32
board = helvepic32_smd

; change microcontroller
board_build.mcu = 32MX250F128D

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Debugging**

`Debugging` currently does not support HelvePic32 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**HelvePic32 MX270**

**Contents**

- `HelvePic32 MX270`
  - `Hardware`
  - `Configuration`
Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX270F256B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>244KB</td>
</tr>
<tr>
<td>RAM</td>
<td>62KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BOXTEC</td>
</tr>
</tbody>
</table>

Configuration

Please use `helvepic32_mx270` ID for *board* option in “`platformio.ini` (Project Configuration File)“:

```plaintext
[env:helvepic32_mx270]
platform = microchippic32
board = helvepic32_mx270
```

You can override default HelvePic32 MX270 settings per build environment using *board_*** option, where *** is a JSON object path from board manifest `helvepic32_mx270.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:helvepic32_mx270]
platform = microchippic32
board = helvepic32_mx270

; change microcontroller
board_build.mcu = 32MX270F256B

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

*Debugging* currently does not support HelvePic32 MX270 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
HelvePic32 Robot

Contents

- HelvePic32 Robot
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX270F256D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>244KB</td>
</tr>
<tr>
<td>RAM</td>
<td>62KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BOXTEC</td>
</tr>
</tbody>
</table>

Configuration

Please use helvepic32_robot ID for *board* option in "platformio.ini" (Project Configuration File):

```
[env:helvepic32_robot]
platform = microchippic32
board = helvepic32_robot
```

You can override default HelvePic32 Robot settings per build environment using *board_*** option, where *** is a JSON object path from board manifest helvepic32_robot.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:helvepic32_robot]
platform = microchippic32
board = helvepic32_robot

; change microcontroller
board_build.mcu = 32MX270F256D

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

*Debugging* currently does not support HelvePic32 Robot board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

HelvePic32 SMD MX270

Contents

- HelvePic32 SMD MX270
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX270F256D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>244KB</td>
</tr>
<tr>
<td>RAM</td>
<td>62KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BOXTEC</td>
</tr>
</tbody>
</table>

Configuration

Please use helvepic32_smd_mx270 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:helvepic32_smd_mx270]
platform = microchippic32
board = helvepic32_smd_mx270
```

You can override default HelvePic32 SMD MX270 settings per build environment using board_*** option, where *** is a JSON object path from board manifest helvepic32_smd_mx270.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:helvepic32_smd_mx270]
platform = microchippic32
board = helvepic32_smd_mx270

; change microcontroller
```

(continues on next page)
board_build.mcu = 32MX270F256D

; change MCU frequency
board_build.f_cpu = 48000000L

Debugging

*Debugging* currently does not support HelvePic32 SMD MX270 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

MikroElektronika Clicker 2

**Contents**

- *MikroElektronika Clicker 2*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high-performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX460F512L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MikroElektronika</td>
</tr>
</tbody>
</table>

Configuration

Please use `clicker2` ID for `board` option in “platformio.ini” (*Project Configuration File)*:
You can override default MikroElektronika Clicker 2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `clicker2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:clicker2]
platform = microchippic32
board = clicker2

; change microcontroller
board_build.mcu = 32MX460F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

## Debugging

*Debugging* currently does not support MikroElektronika Clicker 2 board.

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## MikroElektronika Flip N Click MZ

### Contents

- *MikroElektronika Flip N Click MZ*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

## Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
### Configuration

Please use `flipnclickmz` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:flipnclickmz]
platform = microchippic32
board = flipnclickmz

; change microcontroller
board_build.mcu = 32MZ2048EFH100

; change MCU frequency
board_build.f_cpu = 252000000L
```

You can override default MikroElektronika Flip N Click MZ settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `flipnclickmz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

### Debugging

*Debugging* currently does not support MikroElektronika Flip N Click MZ board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Mini 2.0

**Contents**

- *Mini 2.0*
  - *Hardware*
  - *Configuration*
Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX270F256D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>240KB</td>
</tr>
<tr>
<td>RAM</td>
<td>62KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Fubarino</td>
</tr>
</tbody>
</table>

Configuration

Please use `fubarino_mini_20` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:fubarino_mini_20]
platform = microchippic32
board = fubarino_mini_20
```

You can override default Mini 2.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `fubarino_mini_20.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:fubarino_mini_20]
platform = microchippic32
board = fubarino_mini_20

; change microcontroller
board_build.mcu = 32MX270F256D

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

*Debugging* currently does not support Mini 2.0 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Olimex PIC32-PINGUINO

Contents

- Olimex PIC32-PINGUINO
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX440F256H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>252KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Olimex</td>
</tr>
</tbody>
</table>

Configuration

Please use `pinguino32` ID for `board` option in "`platformio.ini`" *Project Configuration File*:

```
[env:pinguino32]
platform = microchippic32
board = pinguino32
```

You can override default Olimex PIC32-PINGUINO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `pinguino32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pinguino32]
platform = microchippic32
board = pinguino32

; change microcontroller
board_build.mcu = 32MX440F256H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

*Debugging* currently does not support Olimex PIC32-PINGUINO board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

OpenBCI 32bit

Contents

- OpenBCI 32bit
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>32MX250F128B</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
<td>OpenBCI</td>
</tr>
</tbody>
</table>

Configuration

Please use openbci ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:openbci]
platform = microchippic32
board = openbci
```

You can override default OpenBCI 32bit settings per build environment using board_*** option, where *** is a JSON object path from board manifest openbci.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:openbci]
platform = microchippic32
board = openbci

; change microcontroller
```

(continues on next page)
board_build.mcu = 32MX250F128B
; change MCU frequency
board_build.f_cpu = 40000000L

Debugging

*Debugging* currently does not support OpenBCI 32bit board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

PONTECH UAV100

Contents

- **PONTECH UAV100**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX440F512H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PONTECH</td>
</tr>
</tbody>
</table>

Configuration

Please use `usbono_pic32` ID for *board* option in "platformio.ini" *(Project Configuration File)*:
You can override default PONTECH UAV100 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `usbono_pic32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```text
[env:usbono_pic32]
platform = microchippic32
board = usbono_pic32

; change microcontroller
board_build.mcu = 32MX440F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

*Debugging* currently does not support PONTECH UAV100 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Pic32 CUI32-Development Stick

#### Contents

- **Pic32 CUI32-Development Stick**
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

#### Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
### Configuration

Please use `cui32` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:cui32]
platform = microchippic32
board = cui32

; change microcontroller
board_build.mcu = 32MX440F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

You can override default Pic32 CUI32-Development Stick settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cui32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

### Debugging

*Debugging* currently does not support Pic32 CUI32-Development Stick board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Pontech NoFire

#### Contents

- *Pontech NoFire*
  - Hardware
  - Configuration
Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MZ2048EFG100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>200MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.98MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Pontech</td>
</tr>
</tbody>
</table>

Configuration

Please use `nofire` ID for `board` option in “platformio.ini” (*Project Configuration File*):

```ini
[env:nofire]
platform = microchippic32
board = nofire
```

You can override default Pontech NoFire settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nofire.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nofire]
platform = microchippic32
board = nofire

; change microcontroller
board_build.mcu = 32MZ2048EFG100

; change MCU frequency
board_build.f_cpu = 200000000L
```

Debugging

*Debugging* currently does not support Pontech NoFire board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
## Pontech Quick240

### Contents

- Pontech Quick240
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX795F512L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Pontech</td>
</tr>
</tbody>
</table>

### Configuration

Please use `quick240_usb` ID for `board` option in "`platformio.ini`" *(Project Configuration File)*:

```ini
[env:quick240_usb]
platform = microchippic32
board = quick240_usb
```

You can override default Pontech Quick240 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `quick240_usb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:quick240_usb]
platform = microchippic32
board = quick240_usb

; change microcontroller
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

*Debugging* currently does not support Pontech Quick240 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

RGB Station

Contents

- RGB Station
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>32MX270F256D</td>
<td>48MHz</td>
<td>240KB</td>
<td>62KB</td>
<td>ChipKIT</td>
</tr>
</tbody>
</table>

Configuration

Please use rgb_station ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:rgb_station]
platform = microchippic32
board = rgb_station
```

You can override default RGB Station settings per build environment using board_*** option, where *** is a JSON object path from board manifest rgb_station.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:rgb_station]
platform = microchippic32
board = rgb_station

; change microcontroller
board_build.mcu = 32MX270F256D
```

(continues on next page)
Debugging

*Debugging* currently does not support RGB Station board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

SeeedStudio CUI32stem

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>32MX795F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>

Configuration

Please use `cui32stem` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default SeeedStudio CUI32stem settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cui32stem.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:cui32stem]
platform = microchippic32
board = cui32stem

; change microcontroller
board_build.mcu = 32MX795F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

*Debugging* currently does not support SeeedStudio CUI32stem board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### UBW32 MX460

#### Contents

- UBW32 MX460
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!
### Configuration

Please use `ubw32_mx460` ID for board option in "platformio.ini" (Project Configuration File):

```
[env:ubw32_mx460]
platform = microchippic32
board = ubw32_mx460
```

You can override default UBW32 MX460 settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `ubw32_mx460.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ubw32_mx460]
platform = microchippic32
board = ubw32_mx460

; change microcontroller
board_build.mcu = 32MX460F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

*Debugging* currently does not support UBW32 MX460 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### UBW32 MX795

**Contents**

- UBW32 MX795
  - Hardware
  - Configuration
Hardware

Platform **Microchip PIC32**: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX795F512L</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>508KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>UBW32</td>
</tr>
</tbody>
</table>

Configuration

Please use `ubw32_mx795` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[env:ubw32_mx795]
platform = microchippic32
board = ubw32_mx795

; change microcontroller
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

*Debugging* currently does not support UBW32 MX795 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
chipKIT Lenny

Hardware

Platform Microchip PIC32: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MX270F256D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>40MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>chipKIT</td>
</tr>
</tbody>
</table>

Configuration

Please use lenny ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:lenny]
platform = microchippic32
board = lenny
```

You can override default chipKIT Lenny settings per build environment using board_{***} option, where *** is a JSON object path from board manifest lenny.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:lenny]
platform = microchippic32
board = lenny

; change microcontroller
board_build.mcu = 32MX270F256D

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

Debugging currently does not support chipKIT Lenny board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

chipKIT WiFire rev. C

Contents

- chipKIT WiFire rev. C
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *Microchip PIC32*: Microchip’s 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>32MZ2048EFH100</td>
<td>200MHz</td>
<td>1.98MB</td>
<td>512KB</td>
<td>Digilent</td>
</tr>
</tbody>
</table>

Configuration

Please use `chipkit_wifire_revc` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:chipkit_wifire_revc]
platform = microchippic32
board = chipkit_wifire_revc
```

You can override default chipKIT WiFire rev. C settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `chipkit_wifire_revc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:chipkit_wifire_revc]
platform = microchippic32
board = chipkit_wifire_revc
```

; change microcontroller
board_build.mcu = 32M22048EFG100

; change MCU frequency
board_build.f_cpu = 200000000L

Debugging

Debugging currently does not support chipKIT WiFire rev. C board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

1.12.19 Nordic nRF51

BBC micro:bit

Contents

- BBC micro:bit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BBC</td>
</tr>
</tbody>
</table>
**Configuration**

Please use `bbcmicrobit` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:bbcmicrobit]
platform = nordicnrf51
board = bbcmicrobit
```

You can override default BBC micro:bit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `bbcmicrobit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:bbcmicrobit]
platform = nordicnrf51
board = bbcmicrobit

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Uploading**

BBC micro:bit supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:bbcmicrobit]
platform = nordicnrf51
board = bbcmicrobit

upload_protocol = cmsis-dap
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

BBC micro:bit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
CMSIS-DAP | Yes | Yes
J-LINK |  |  

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

BluzDK

Contents

- BluzDK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BluzDK</td>
</tr>
</tbody>
</table>

Configuration

Please use bluz dk ID for board option in “platformio.ini” (Project Configuration File):
You can override default BlzDK settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bluz_dk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:bluz_dk]
platform = nordicnrf51
board = bluz_dk

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Uploading

BlzDK supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:bluz_dk]
platform = nordicnrf51
board = bluz_dk

upload_protocol = jlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

BlzDK does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Calliope mini

Contents

- **Calliope mini**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Calliope</td>
</tr>
</tbody>
</table>

Configuration

Please use `calliope_mini` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:calliope_mini]
platform = nordicnrf51
board = calliope_mini
```
You can override default Calliope mini settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `calliope_mini.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:calliope_mini]
platform = nordicnrf51
board = calliope_mini

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Uploading

Calliope mini supports the next uploading protocols:

- `cmsis-dap`
- `jlink`
- `mbed`

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:calliope_mini]
platform = nordicnrf51
board = calliope_mini

upload_protocol = cmsis-dap
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Calliope mini has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Delta DFCM-NNN40

Contents

- Delta DFCM-NNN40
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Delta</td>
</tr>
</tbody>
</table>

Configuration

Please use dfcm_nnn40 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:dfcm_nnn40]
platform = nordicnrf51
board = dfcm_nnn40
```

You can override default Delta DFCM-NNN40 settings per build environment using board_*** option, where *** is a JSON object path from board manifest dfcm_nnn40.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:dfcm_nnn40]
platform = nordicnrf51
board = dfcm_nnn40
```

(continues on next page)
Uploading

Delta DFCM-NNN40 supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```
[env:dfcm_nnn40]
platform = nordicnrf51
board = dfcm_nnn40
upload_protocol = cmsis-dap
```
Delta DFCM-NNN50

Contents

- Delta DFCM-NNN50
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Delta</td>
</tr>
</tbody>
</table>

Configuration

Please use `delta_dfcm_nnn50` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:delta_dfcm_nnn50]
platform = nordicnrf51
board = delta_dfcm_nnn50
```

You can override default Delta DFCM-NNN50 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `delta_dfcm_nnn50.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:delta_dfcm_nnn50]
platform = nordicnrf51
board = delta_dfcm_nnn50

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```
Uploading

Delta DFCM-NNN50 supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```
[env:delta_dfcm_nnn50]
platform = nordicnrf51
board = delta_dfcm_nnn50
upload_protocol = cmsis-dap
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

Delta DFCM-NNN50 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

```
Compatible Tools  On-board  Default
CMSIS-DAP         Yes       Yes
```

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

JKSoft Wallbot BLE

Contents

- JKSoft Wallbot BLE
  - Hardware
  - Configuration
Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>JKSoft</td>
</tr>
</tbody>
</table>

Configuration

Please use wallbot_ble ID for board option in “platformio.ini” (Project Configuration File):

```
[env:wallbot_ble]
platform = nordicnrf51
board = wallbot_ble
```

You can override default JKSoft Wallbot BLE settings per build environment using board_*** option, where *** is a JSON object path from board manifest wallbot_ble.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:wallbot_ble]
platform = nordicnrf51
board = wallbot_ble

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

JKSoft Wallbot BLE supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:
[env:wallbot_ble]
platform = nordicnrf51
board = wallbot_ble
upload_protocol = cmsis-dap

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

JKSoft Wallbot BLE has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Nordic Beacon Kit (PCA20006)

Contents

- *Nordic Beacon Kit (PCA20006)*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nordic</td>
</tr>
</tbody>
</table>

Configuration

Please use nrf51_beacon ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nrf51_beacon]
platform = nordicnrf51
board = nrf51_beacon
```

You can override default Nordic Beacon Kit (PCA20006) settings per build environment using board_*** option, where *** is a JSON object path from board manifest nrf51_beacon.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nrf51_beacon]
platform = nordicnrf51
board = nrf51_beacon

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Nordic Beacon Kit (PCA20006) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```
[env:nrf51_beacon]
platform = nordicnrf51
board = nrf51_beacon
```

(continues on next page)
upload_protocol = jlink

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Nordic Beacon Kit (PCA20006) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Nordic nRF51 Dongle (PCA10031)**

**Contents**

- **Nordic nRF51 Dongle (PCA10031)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nordic</td>
</tr>
</tbody>
</table>

Configuration

Please use nrf51_dongle ID for *board* option in “platformio.ini” (Project Configuration File):

```
[env:nrf51_dongle]
platform = nordicnrf51
board = nrf51_dongle
```

You can override default Nordic nRF51 Dongle (PCA10031) settings per build environment using *board_*** option, where *** is a JSON object path from board manifest nrf51_dongle.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nrf51_dongle]
platform = nordicnrf51
board = nrf51_dongle

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Nordic nRF51 Dongle (PCA10031) supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed
- nrfjprog

Default protocol is jlink

You can change upload protocol using *upload_protocol* option:

```
[env:nrf51_dongle]
platform = nordicnrf51
board = nrf51_dongle

upload_protocol = jlink
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Nordic nRF51 Dongle (PCA10031) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the 'things' in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Nordic nRF51822-mKIT

**Contents**

- **Nordic nRF51822-mKIT**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including
Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nordic</td>
</tr>
</tbody>
</table>

Configuration

Please use nrf51_mkit ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nrf51_mkit]
platform = nordicnrf51
board = nrf51_mkit
```

You can override default Nordic nRF51822-mKIT settings per build environment using board_*** option, where *** is a JSON object path from board manifest nrf51_mkit.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nrf51_mkit]
platform = nordicnrf51
board = nrf51_mkit

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

Nordic nRF51822-mKIT supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```ini
[env:nrf51_mkit]
platform = nordicnrf51
board = nrf51_mkit

upload_protocol = cmsis-dap
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Nordic nRF51822-mKIT has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Nordic nRF51X22 Development Kit(PCA1000X)

Contents

- Nordic nRF51X22 Development Kit(PCA1000X)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nordic</td>
</tr>
</tbody>
</table>
Configuration

Please use nrf51_dk ID for *board* option in “platformio.ini” (Project Configuration File):

``` ini
[env:nrf51_dk]
platform = nordicnrf51
board = nrf51_dk
```

You can override default Nordic nRF51X22 Development Kit(PCA1000X) settings per build environment using *board_*** option, where *** is a JSON object path from board manifest nrf51_dk.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

``` ini
[env:nrf51_dk]
platform = nordicnrf51
board = nrf51_dk

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Nordic nRF51X22 Development Kit(PCA1000X) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is *jlink*

You can change upload protocol using *upload_protocol* option:

``` ini
[env:nrf51_dk]
platform = nordicnrf51
board = nrf51_dk

upload_protocol = jlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging tools using `debug_tool` option in “platformio.ini” (Project Configuration File).

Nordic nRF51X22 Development Kit(PCA1000X) has on-board debug probe and **is ready** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephy RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

## OSHChip

### Contents

- **OSHChip**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>OSHChip</td>
</tr>
</tbody>
</table>
Configuration

Please use oshchip ID for board option in “platformio.ini” (Project Configuration File):

```
[env:oshchip]
platform = nordicnrf51
board = oshchip
```

You can override default OSHChip settings per build environment using board_*** option, where *** is a JSON object path from board manifest oshchip.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:oshchip]
platform = nordicnrf51
board = oshchip

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

OSHChip supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```
[env:oshchip]
platform = nordicnrf51
board = oshchip

upload_protocol = jlink
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

OSHChip does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | |
J-LINK | | |
ST-LINK | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

RedBearLab BLE Nano 1.5

Contents

- RedBearLab BLE Nano 1.5
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RedBearLab</td>
</tr>
</tbody>
</table>

Configuration

Please use **redBearLabBLENano** ID for `board` option in “platformio.ini” (Project Configuration File):

```plaintext
[env:redBearLabBLENano]
platform = nordinrf51
board = redBearLabBLENano
```
You can override default RedBearLab BLE Nano 1.5 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `redBearLabBLENano.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:redBearLabBLENano]
platform = nordicnrf51
board = redBearLabBLENano

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Uploading

RedBearLab BLE Nano 1.5 supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `jlink`
- `mbed`
- `nrfjprog`
- `stlink`

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:redBearLabBLENano]
platform = nordicnrf51
board = redBearLabBLENano

upload_protocol = cmsis-dap
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

RedBearLab BLE Nano 1.5 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe |  |  
CMSIS-DAP | Yes | Yes
J-LINK |  |  
ST-LINK |  |  

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the 'things' in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

RedBearLab nRF51822

Contents

- RedBearLab nRF51822
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
<td>RedBearLab</td>
</tr>
</tbody>
</table>
Configuration

Please use redBearLab ID for board option in “platformio.ini” (Project Configuration File):

```
[env:redBearLab]
platform = nordicnrf51
board = redBearLab
```

You can override default RedBearLab nRF51822 settings per build environment using board_*** option, where *** is a JSON object path from board manifest redBearLab.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:redBearLab]
platform = nordicnrf51
board = redBearLab

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

RedBearLab nRF51822 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```
[env:redBearLab]
platform = nordicnrf51
board = redBearLab

upload_protocol = cmsis-dap
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

RedBearLab nRF51822 has on-board debug probe and is ready for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Seeed Arch BLE

**Contents**

- Seeed Arch BLE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Nordic nRF51*: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>
Configuration

Please use `seeedArchBLE` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:seeedArchBLE]
platform = nordicnrf51
board = seeedArchBLE
```

You can override default Seeed Arch BLE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `seeedArchBLE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:seeedArchBLE]
platform = nordicnrf51
board = seeedArchBLE

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

Seeed Arch BLE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:seeedArchBLE]
platform = nordicnrf51
board = seeedArchBLE

upload_protocol = cmsis-dap
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Seeed Arch BLE has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Seeed Arch Link

- **Seeed Arch Link**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>
Configuration

Please use `seeedArchLink ID` for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:seeedArchLink]
platform = nordicnrf51
board = seeedArchLink
```

You can override default Seeed Arch Link settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `seeedArchLink.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:seeedArchLink]
platform = nordicnrf51
board = seeedArchLink

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

Seeed Arch Link supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:seeedArchLink]
platform = nordicnrf51
board = seeedArchLink

upload_protocol = cmsis-dap
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” *(Project Configuration File)*.

Seeed Arch Link has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

#### Seeed Tiny BLE

**Contents**

- Seeed Tiny BLE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>
Configuration

Please use `seeedTinyBLE` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:seeedTinyBLE]
platform = nordicnrf51
board = seeedTinyBLE
```

You can override default Seeed Tiny BLE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `seeedTinyBLE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:seeedTinyBLE]
platform = nordicnrf51
board = seeedTinyBLE

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

Seeed Tiny BLE supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `jlink`
- `mbed`
- `nrfjprog`
- `stlink`

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:seeedTinyBLE]
platform = nordicnrf51
board = seeedTinyBLE

upload_protocol = cmsis-dap
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Seeed Tiny BLE has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Sino:Bit

Contents

- Sino:Bit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF51822</td>
<td>sino:bit</td>
</tr>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
</tbody>
</table>
Configuration

Please use Sinobit ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:Sinobit]
platform = nordicnrf51
board = Sinobit
```

You can override default Sino:Bit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest Sinobit.json. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:Sinobit]
platform = nordicnrf51
board = Sinobit

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Sino:Bit supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:Sinobit]
platform = nordicnrf51
board = Sinobit

upload_protocol = jlink
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sino:Bit does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

## Switch Science mbed HRM1017

### Contents

- Switch Science mbed HRM1017
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform **Nordic nRF51**: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Switch Science</td>
</tr>
</tbody>
</table>

## Configuration

Please use `hrm1017` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:hrm1017]
platform = nvidia
board = hrm1017
```
You can override default Switch Science mbed HRM1017 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hrm1017.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:hrm1017]
platform = nordicnrf51
board = hrm1017

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Uploading**

Switch Science mbed HRM1017 supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:hrm1017]
platform = nordicnrf51
board = hrm1017

upload_protocol = cmsis-dap
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.

Switch Science mbed HRM1017 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1.12. Boards
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Switch Science mbed TY51822r3

Contents

- Switch Science mbed TY51822r3
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Switch Science</td>
</tr>
</tbody>
</table>

Configuration

Please use ty51822r3 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ty51822r3]
platform = nordicnrf51
board = ty51822r3
```

You can override default Switch Science mbed TY51822r3 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ty51822r3.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ty51822r3]
platform = nordicnrf51
board = ty51822r3
```

(continues on next page)
; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L

**Uploading**

Switch Science mbed TY51822r3 supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:ty51822r3]
platform = nordicnrf51
board = ty51822r3
upload_protocol = cmsis-dap
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Switch Science mbed TY51822r3 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>
VNG VBLUNO51

Contents

- VNG VBLUNO51
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>VNG</td>
</tr>
</tbody>
</table>

Configuration

Please use vbluno51 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:vbluno51]
platform = nordicnrf51
board = vbluno51
```

You can override default VNG VBLUNO51 settings per build environment using board_*** option, where *** is a JSON object path from board manifest vbluno51.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:vbluno51]
platform = nordicnrf51
board = vbluno51

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```
Uploading

VNG VBLUNO51 supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using `upload_protocol` option:

```
[env:vbluno51]
platform = nordicnrf51
board = vbluno51
upload_protocol = cmsis-dap
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

VNG VBLUNO51 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Waveshare BLE400

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Waveshare BLE400</td>
</tr>
</tbody>
</table>
Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Waveshare</td>
</tr>
</tbody>
</table>

Configuration

Please use waveshare_ble400 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:waveshare_ble400]
platform = nordicnrf51
board = waveshare_ble400
```

You can override default Waveshare BLE400 settings per build environment using board_*** option, where *** is a JSON object path from board manifest waveshare_ble400.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:waveshare_ble400]
platform = nordicnrf51
board = waveshare_ble400

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Waveshare BLE400 supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
• stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```ini
[env:waveshare_ble400]
platform = nordicnrf51
board = waveshare_ble400
upload_protocol = jlink
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using debug_tool option in “platformio.ini” (Project Configuration File).

Waveshare BLE400 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### ng-beacon

**Contents**

- ng-beacon
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ng-beacon</td>
</tr>
</tbody>
</table>

Configuration

Please use ng_beacon ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ng_beacon]
platform = nordicnrf51
board = ng_beacon
```

You can override default ng-beacon settings per build environment using board_*** option, where *** is a JSON object path from board manifest ng_beacon.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ng_beacon]
platform = nordicnrf51
board = ng_beacon

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

ng-beacon supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:
[env:ng_beacon]
platform = nordicnrf51
board = ng_beacon
upload_protocol = jlink

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ng-beacon does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

Arduino

Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

y5 nRF51822 mbug

Contents

- y5 nRF51822 mbug
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform Nordic nRF51: The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF51822</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>y5 design</td>
</tr>
</tbody>
</table>

Configuration

Please use nrf51822_y5_mbug ID for board option in "platformio.ini" (Project Configuration File):

```
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug
```

You can override default y5 nRF51822 mbug settings per build environment using board_*** option, where *** is a JSON object path from board manifest nrf51822_y5_mbug.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

y5 nRF51822 mbug supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug
upload_protocol = cmsis-dap

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

y5 nRF51822 mbug has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

1.12.20 Nordic nRF52

96Boards Nitrogen

### Contents

- 96Boards Nitrogen
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>96Boards</td>
</tr>
</tbody>
</table>

Configuration

Please use 96b_nitrogen ID for board option in “platformio.ini” (Project Configuration File):

```
[env:96b_nitrogen]
platform = nordicnrf52
board = 96b_nitrogen
```

You can override default 96Boards Nitrogen settings per build environment using board_*** option, where *** is a JSON object path from board manifest 96b_nitrogen.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:96b_nitrogen]
platform = nordicnrf52
board = 96b_nitrogen

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

96Boards Nitrogen supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```
[env:96b_nitrogen]
platform = nordicnrf52
board = 96b_nitrogen

upload_protocol = jlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

96Boards Nitrogen does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
<td></td>
</tr>
</tbody>
</table>

Adafruit Bluefruit nRF52832 Feather

Contents

- Adafruit Bluefruit nRF52832 Feather
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

### Configuration

Please use `adafruit_feather_nrf52832` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_feather_nrf52832]
platform = nordicnrf52
board = adafruit_feather_nrf52832
```

You can override default Adafruit Bluefruit nRF52832 Feather settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_feather_nrf52832.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_feather_nrf52832]
platform = nordicnrf52
board = adafruit_feather_nrf52832

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Adafruit Bluefruit nRF52832 Feather supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_feather_nrf52832]
platform = nordicnrf52
board = adafruit_feather_nrf52832

upload_protocol = nrfutil
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Bluefruit nRF52832 Feather does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Adafruit CLUE nRF52840

Contents

- Adafruit CLUE nRF52840
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
### Configuration

Please use `adafruit_clue_nrf52840` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_clue_nrf52840]
platform = nordicnrf52
board = adafruit_clue_nrf52840
```

You can override default Adafruit CLUE nRF52840 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_clue_nrf52840.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_clue_nrf52840]
platform = nordicnrf52
board = adafruit_clue_nrf52840

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Adafruit CLUE nRF52840 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_clue_nrf52840]
platform = nordicnrf52
board = adafruit_clue_nrf52840

upload_protocol = nrfutil
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit CLUE nRF52840 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Adafruit Feather Bluefruit Sense

Contents

- Adafruit Feather Bluefruit Sense
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
### Configuration

Please use `adafruit_feather_nrf52840_sense` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:adafruit_feather_nrf52840_sense]
platform = nordicnrf52
board = adafruit_feather_nrf52840_sense
```

You can override default Adafruit Feather Bluefruit Sense settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_feather_nrf52840_sense.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_feather_nrf52840_sense]
platform = nordicnrf52
board = adafruit_feather_nrf52840_sense

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Adafruit Feather Bluefruit Sense supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_feather_nrf52840_sense]
platform = nordicnrf52
board = adafruit_feather_nrf52840_sense

upload_protocol = nrfutil
```
**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Feather Bluefruit Sense does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Adafruit Feather nRF52840 Express**

**Contents**

- *Adafruit Feather nRF52840 Express*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `adafruit_feather_nrf52840` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:adafruit_feather_nrf52840]
platform = nordicnrf52
board = adafruit_feather_nrf52840
```

You can override default Adafruit Feather nRF52840 Express settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_feather_nrf52840.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_feather_nrf52840]
platform = nordicnrf52
board = adafruit_feather_nrf52840

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

**Uploading**

Adafruit Feather nRF52840 Express supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_feather_nrf52840]
platform = nordicnrf52
board = adafruit_feather_nrf52840

upload_protocol = nrfutil
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Adafruit Feather nRF52840 Express does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Arduino Nano 33 BLE

Contents

- Arduino Nano 33 BLE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
Microcontroller  | NRF52840  
---|---
Frequency  | 64MHz  
Flash  | 960KB  
RAM  | 256KB  
Vendor  | Arduino

**Configuration**

Please use `nano33ble` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:nano33ble]
platform = nordicnrf52
board = nano33ble
```

You can override default Arduino Nano 33 BLE settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nano33ble.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nano33ble]
platform = nordicnrf52
board = nano33ble

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

**Uploading**

Arduino Nano 33 BLE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- sam-ba

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nano33ble]
platform = nordicnrf52
board = nano33ble

upload_protocol = sam-ba
```
### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arduino Nano 33 BLE does not have on-board debug probe and is NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### BBC micro:bit V2

**Contents**

- **BBC micro:bit V2**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52833</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>BBC</td>
</tr>
</tbody>
</table>

### Configuration

Please use `bbcmicrobit_v2` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:bbcmicrobit_v2]
platform = nordicnrf52
board = bbcmicrobit_v2
```

You can override default BBC micro:bit V2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bbcmicrobit_v2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:bbcmicrobit_v2]
platform = nordicnrf52
board = bbcmicrobit_v2

; change microcontroller
board_build.mcu = nrf52833

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

BBC micro:bit V2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:bbcmicrobit_v2]
platform = nordicnrf52
board = bbcmicrobit_v2

upload_protocol = cmsis-dap
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

BBC micro:bit V2 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

BL652 Development Kit

Contents

- **BL652 Development Kit**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
### Configuration

Please use `laird_bl652_dvk` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:laird_bl652_dvk]
platform = nordicnrf52
board = laird_bl652_dvk
```

You can override default BL652 Development Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `laird_bl652_dvk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:laird_bl652_dvk]
platform = nordicnrf52
board = laird_bl652_dvk

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

BL652 Development Kit supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:laird_bl652_dvk]
platform = nordicnrf52
board = laird_bl652_dvk

upload_protocol = jlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

BL652 Development Kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

BL654 Development Kit

Contents

- BL654 Development Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
### PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th><strong>Microcontroller</strong></th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frequency</strong></td>
<td>64MHz</td>
</tr>
<tr>
<td><strong>Flash</strong></td>
<td>1MB</td>
</tr>
<tr>
<td><strong>RAM</strong></td>
<td>256KB</td>
</tr>
<tr>
<td><strong>Vendor</strong></td>
<td>Laird Connectivity</td>
</tr>
</tbody>
</table>

#### Configuration

Please use laird_bl654_dvk ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:laird_bl654_dvk]
platform = nordicnrf52
board = laird_bl654_dvk
```

You can override default BL654 Development Kit settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `laird_bl654_dvk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:laird_bl654_dvk]
platform = nordicnrf52
board = laird_bl654_dvk

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

#### Uploading

BL654 Development Kit supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:laird_bl654_dvk]
platform = nordicnrf52
board = laird_bl654_dvk

upload_protocol = jlink
```
Debugging

"1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in `platformio.ini` (Project Configuration File).

BL654 Development Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Bluey nRF52832 IoT

**Contents**

- Bluey nRF52832 IoT
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Nordic nRF52:** The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.
**Microcontroller** | **NRF52832**  
---|---  
**Frequency** | **64MHz**  
**Flash** | **512KB**  
**RAM** | **64KB**  
**Vendor** | **Electronut Labs**

**Configuration**

Please use bluey ID for *board* option in “*platformio.ini* (Project Configuration File):

```
[env:bluey]
platform = nordicnrf52
board = bluey
```

You can override default Bluey nRF52832 IoT settings per build environment using *board_*** option, where *** is a JSON object path from board manifest *bluey.json*. For example, *board_build.mcu, board_build.f_cpu*, etc.

```
[env:bluey]
platform = nordicnrf52
board = bluey

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

**Uploading**

Bluey nRF52832 IoT supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is *jlink*

You can change upload protocol using *upload_protocol* option:

```
[env:bluey]
platform = nordicnrf52
board = bluey

upload_protocol = jlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Bluey nRF52832 IoT does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Circuit Playground Bluefruit

Contents

- Circuit Playground Bluefruit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>
Configuration

Please use `adafruit_cplaynrf52840` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:adafruit_cplaynrf52840]
platform = nordicnrf52
board = adafruit_cplaynrf52840
```

You can override default Circuit Playground Bluefruit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_cplaynrf52840.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_cplaynrf52840]
platform = nordicnrf52
board = adafruit_cplaynrf52840

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Circuit Playground Bluefruit supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_cplaynrf52840]
platform = nordicnrf52
board = adafruit_cplaynrf52840

upload_protocol = nrfutil
```

Debugging

*Debugging* - "1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Circuit Playground Bluefruit does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Delta DFBM-NQ620

Contents

- Delta DFBM-NQ620
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Delta</td>
</tr>
</tbody>
</table>
PlatformIO Documentation, Release 5.0.5a1

Configuration

Please use delta_dfbm_nq620 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:delta_dfbm_nq620]
platform = nordicnrf52
board = delta_dfbm_nq620
```

You can override default Delta DFBM-NQ620 settings per build environment using board_*** option, where *** is a JSON object path from board manifest delta_dfbm_nq620.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:delta_dfbm_nq620]
platform = nordicnrf52
board = delta_dfbm_nq620

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Delta DFBM-NQ620 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```
[env:delta_dfbm_nq620]
platform = nordicnrf52
board = delta_dfbm_nq620

upload_protocol = cmsis-dap
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging Tools & Debug Probes using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

Delta DFBM-NQ620 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**ElectronutLabs Blip**

**Contents**

- ElectronutLabs Blip
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ElectronutLabs</td>
</tr>
</tbody>
</table>
Configuration

Please use `electronut_blip` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:electronut_blip]
platform = nordicnrf52
board = electronut_blip
```

You can override default ElectronutLabs Blip settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `electronut_blip.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:electronut_blip]
platform = nordicnrf52
board = electronut_blip

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

ElectronutLabs Blip supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `blackmagic`

You can change upload protocol using `upload_protocol` option:

```
[env:electronut_blip]
platform = nordicnrf52
board = electronut_blip

upload_protocol = blackmagic
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
ElectronutLabs Blip has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

**ElectronutLabs Papyr**

**Contents**

- ElectronutLabs Papyr
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ElectronutLabs</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `electronut_papyr ID` for `board` option in "platformio.ini" (Project Configuration File):
You can override default ElectronutLabs Papyr settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `electronut_papyr.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:electronut_papyr]
platform = nordicnrf52
board = electronut_papyr

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

**Uploading**

ElectronutLabs Papyr supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `blackmagic`

You can change upload protocol using `upload_protocol` option:

```ini
[env:electronut_papyr]
platform = nordicnrf52
board = electronut_papyr

upload_protocol = blackmagic
```

**Debugging**

> **Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

ElectronutLabs Papyr has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Holyiot YJ-16019

**Contents**

- **Holyiot YJ-16019**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Holyiot</td>
</tr>
</tbody>
</table>

### Configuration

Please use `holyiot_yj16019` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:holyiot_yj16019]
platform = nordicnrf52
board = holyiot_yj16019
```
You can override default Holyiot YJ-16019 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `holyiot_yj16019.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:holyiot_yj16019]
platform = nordicnrf52
board = holyiot_yj16019

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Holyiot YJ-16019 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:holyiot_yj16019]
platform = nordicnrf52
board = holyiot_yj16019

upload_protocol = jlink
```

### Debugging

_Debbuging_ - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging _Tools & Debug Probes_ using `debug_tool` option in “platformio.ini” (_Project Configuration File_).

Holyiot YJ-16019 does not have on-board debug probe and is NOT READY for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | 
CMSIS-DAP | 
J-LINK | 
ST-LINK | 

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

ItsyBitsy nRF52840 Express

Contents
- ItsyBitsy nRF52840 Express
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_itsybitsy_nrf52840` ID for `board` option in “platformio.ini” (Project Configuration File):

```python
[env:adafruit_itsybitsy_nrf52840]
platform = nordicnrf52
board = adafruit_itsybitsy_nrf52840
```
You can override default ItsyBitsy nRF52840 Express settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_itsybitsy_nrf52840.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_itsybitsy_nrf52840]
platform = nordicnrf52
board = adafruit_itsybitsy_nrf52840

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

ItsyBitsy nRF52840 Express supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_itsybitsy_nrf52840]
platform = nordicnrf52
board = adafruit_itsybitsy_nrf52840

upload_protocol = nrfutil
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ItsyBitsy nRF52840 Express does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
### PlatformIO Documentation, Release 5.0.5a1

#### Compatible Tools

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

#### Laird Connectivity Pinnacle 100 DVK

#### Contents

- Laird Connectivity Pinnacle 100 DVK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Laird Connectivity</td>
</tr>
</tbody>
</table>

#### Configuration

Please use laird_pinnacle_100_dvk ID for board option in "platformio.ini" (Project Configuration File):

```plaintext
[env:laird_pinnacle_100_dvk]
platform = nordicnrf52
board = laird_pinnacle_100_dvk
```
You can override default Laird Connectivity Pinnacle 100 DVK settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `laird_pinnacle_100_dvk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:laird_pinnacle_100_dvk]
platform = nordicnrf52
board = laird_pinnacle_100_dvk

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Laird Connectivity Pinnacle 100 DVK supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:laird_pinnacle_100_dvk]
platform = nordicnrf52
board = laird_pinnacle_100_dvk

upload_protocol = jlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Laird Connectivity Pinnacle 100 DVK does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
## Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

## Makerdiary nRF52832-MDK

### Contents

- Makerdiary nRF52832-MDK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Makerdiary</td>
</tr>
</tbody>
</table>

## Configuration

Please use `nrf52832_mdk` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:nrf52832_mdk]
platform = nordicnrf52
board = nrf52832_mdk
```
You can override default Makerdiary nRF52832-MDK settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nrf52832_mdk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nrf52832_mdk]
platform = nordicnrf52
board = nrf52832_mdk

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Makerdiary nRF52832-MDK supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `jlink`
- `mbed`
- `nrfjprog`
- `stlink`

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nrf52832_mdk]
platform = nordicnrf52
board = nrf52832_mdk

upload_protocol = cmsis-dap
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Makerdiary nRF52832-MDK has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
## Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

## Makerdiary nRF52840-MDK

### Contents

- Makerdiary nRF52840-MDK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Makerdiary</td>
</tr>
</tbody>
</table>

### Configuration

Please use `nrf52840_mdk` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nrf52840_mdk]
platform = nordinfnrf52
board = nrf52840_mdk
```
You can override default Makerdiary nRF52840-MDK settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nrf52840_mdk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nrf52840_mdk]
platform = nordicnrf52
board = nrf52840_mdk

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Makerdiary nRF52840-MDK supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:nrf52840_mdk]
platform = nordicnrf52
board = nrf52840_mdk

upload_protocol = cmsis-dap
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

Makerdiary nRF52840-MDK has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe |  | 
CMSIS-DAP | Yes | Yes
J-LINK |  | 
ST-LINK |  | 

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Metro nRF52840 Express

Contents

- Metro nRF52840 Express
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use `adafruit_metro_nrf52840` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:adafruit_metro_nrf52840]
platform = nordicnrf52
board = adafruit_metro_nrf52840
```
You can override default Metro nRF52840 Express settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_metro_nrf52840.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:adafruit_metro_nrf52840]
platform = nordicnrf52
board = adafruit_metro_nrf52840

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

## Uploading

Metro nRF52840 Express supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:adafruit_metro_nrf52840]
platform = nordicnrf52
board = adafruit_metro_nrf52840

upload_protocol = nrfutil
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" *(Project Configuration File)*.

Metro nRF52840 Express does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
---|---|---
*Black Magic Probe* | Yes | |
CMSIS-DAP | | |
J-LINK | | |
ST-LINK | | |

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Nordic Thingy:52 (nRF52-PCA20020)**

**Contents**

- *Nordic Thingy:52 (nRF52-PCA20020)*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
<td>Nordic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `thingy_52` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```plaintext
[env:thingy_52]
platform = nordinnf52
board = thingy_52
```
You can override default Nordic Thingy:52 (nRF52-PCA20020) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `thingy_52.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:thingy_52]
platform = nordicnrf52
board = thingy_52

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Nordic Thingy:52 (nRF52-PCA20020) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:thingy_52]
platform = nordicnrf52
board = thingy_52

upload_protocol = jlink
```

### Debugging

_Debbuging_ - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging _Tools & Debug Probes_ using `debug_tool` option in “platformio.ini” (Project Configuration File).

Nordic Thingy:52 (nRF52-PCA20020) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Nordic nRF52-DK

#### Contents

- Nordic nRF52-DK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
<td>Nordic</td>
</tr>
</tbody>
</table>

#### Configuration

Please use nrf52_dk ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk
```
You can override default Nordic nRF52-DK settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nrf52_dk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

## Uploading

Nordic nRF52-DK supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `jlink`
- `mbed`
- `nrfjprog`
- `stlink`

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk

upload_protocol = jlink
```

## Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

### Warning:
You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Nordic nRF52-DK has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephy RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Nordic nRF52840-DK

#### Contents

- Nordic nRF52840-DK
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nordic</td>
</tr>
</tbody>
</table>
Configuration

Please use nrf52840_dk ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nrf52840_dk]
platform = nordicnrf52
board = nrf52840_dk
```

You can override default Nordic nRF52840-DK settings per build environment using board_*** option, where *** is a JSON object path from board manifest nrf52840_dk.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nrf52840_dk]
platform = nordicnrf52
board = nrf52840_dk

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Nordic nRF52840-DK supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```
[env:nrf52840_dk]
platform = nordicnrf52
board = nrf52840_dk

upload_protocol = jlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Nordic nRF52840-DK has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephy RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

**Nordic nRF52840-DK (Adafruit BSP)**

**Contents**

- Nordic nRF52840-DK (Adafruit BSP)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nordic</td>
</tr>
</tbody>
</table>
Configuration

Please use nrf52840_dk_adafruit ID for board option in “platformio.ini” (Project Configuration File):

```plaintext
[env:nrf52840_dk_adafruit]
platform = nordicnrf52
board = nrf52840_dk_adafruit
```

You can override default Nordic nRF52840-DK (Adafruit BSP) settings per build environment using board_*** option, where *** is a JSON object path from board manifest nrf52840_dk_adafruit.json. For example, board_build.mcu, board_build.f_cpu, etc.

```plaintext
[env:nrf52840_dk_adafruit]
platform = nordicnrf52
board = nrf52840_dk_adafruit

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Nordic nRF52840-DK (Adafruit BSP) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```plaintext
[env:nrf52840_dk_adafruit]
platform = nordicnrf52
board = nrf52840_dk_adafruit

upload_protocol = jlink
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
Nordic nRF52840-DK (Adafruit BSP) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**Particle Argon**

**Contents**

- `Particle Argon`
  - `Hardware`
  - `Configuration`
  - `Uploading`
  - `Debugging`
  - `Frameworks`

**Hardware**

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Particle</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `particle_argon` ID for `board` option in "platformio.ini" *(Project Configuration File)*:
You can override default Particle Argon settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `particle_argon.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:particle_argon]
platform = nordicnrf52
board = particle_argon

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Particle Argon supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```ini
[env:particle_argon]
platform = nordicnrf52
board = particle_argon

upload_protocol = nrfutil
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

Particle Argon does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
## PlatformIO Documentation, Release 5.0.5a1

### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Particle Boron

#### Contents

- **Particle Boron**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Particle</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `particle_boron` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:particle_boron]
platform = nordicnrf52
board = particle_boron
```
You can override default Particle Boron settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `particle_boron.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:particle_boron]
platform = nordicnrf52
board = particle_boron

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Particle Boron supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```
[env:particle_boron]
platform = nordicnrf52
board = particle_boron

upload_protocol = nrfutil
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

Particle Boron does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Particle Xenon

Contents
- Particle Xenon
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Particle</td>
</tr>
</tbody>
</table>

Configuration

Please use particle_xenon ID for board option in “platformio.ini” (Project Configuration File):

```
[env:particle_xenon]
platform = nordichrf52
board = particle_xenon
```

You can override default Particle Xenon settings per build environment using board_*** option, where *** is a JSON object path from board manifest particle_xenon.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:particle_xenon]
platform = nordichrf52
board = particle_xenon
```

(continues on next page)
Uploading

Particle Xenon supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is nrfutil

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:particle_xenon]
platform = nordicnrf52
board = particle_xenon
upload_protocol = nrfutil
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Particle Xenon does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Raytac MDBT50Q-RX Dongle

Contents

- Raytac MDBT50Q-RX Dongle
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>796KB</td>
</tr>
<tr>
<td>RAM</td>
<td>243KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Raytac</td>
</tr>
</tbody>
</table>

Configuration

Please use `raytac_mdbt50q_rx` ID for `board` option in “`platformio.ini` (Project Configuration File):”

```ini
[env:raytac_mdbt50q_rx]
platform = nordicnrf52
board = raytac_mdbt50q_rx
```

You can override default Raytac MDBT50Q-RX Dongle settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `raytac_mdbt50q_rx.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

Raytac MDBT50Q-RX Dongle supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- nrfutil
- stlink

Default protocol is nrfutil

You can change upload protocol using upload_protocol option:

```ini
[env:raytac_mdbt50q_rx]
platform = nordicnrf52
board = raytac_mdbt50q_rx
upload_protocol = nrfutil
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Raytac MDBT50Q-RX Dongle does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

RedBearLab BLE Nano 2

Contents

- RedBearLab BLE Nano 2
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RedBearLab</td>
</tr>
</tbody>
</table>

Configuration

Please use redbear_blenano2 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:redbear_blenano2]
platform = nordicnrf52
board = redbear_blenano2
```

You can override default RedBearLab BLE Nano 2 settings per build environment using board_*** option, where *** is a JSON object path from board manifest redbear_blenano2.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:redbear_blenano2]
platform = nordicnrf52
board = redbear_blenano2
```

(continues on next page)
; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L

Uploading

RedBearLab BLE Nano 2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```
[env:redbear_blenano2]
platform = nordicnrf52
board = redbear_blenano2
upload_protocol = cmsis-dap
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

RedBearLab BLE Nano 2 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

RedBearLab Blend 2

Contents

- RedBearLab Blend 2
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RedBearLab</td>
</tr>
</tbody>
</table>

Configuration

Please use redbear_blend2 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:redbear_blend2]
platform = nrf52
board = redbear_blend2
```
You can override default RedBearLab Blend 2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `redbear_blend2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:redbear_blend2]
platform = nordicnrf52
board = redbear_blend2

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

RedBearLab Blend 2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is **cmsis-dap**

You can change upload protocol using `upload_protocol` option:

```ini
[env:redbear_blend2]
platform = nordicnrf52
board = redbear_blend2

upload_protocol = cmsis-dap
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

RedBearLab Blend 2 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe |  |  
CMSIS-DAP | Yes | Yes
J-LINK |  |  
ST-LINK |  |  

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Ruuvi Tag

Contents

- Ruvi Tag
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Ruvi</td>
</tr>
</tbody>
</table>

Configuration

Please use ruuvitag ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ruuvitag]
platform = nordicnrf52
board = ruuvitag
```
You can override default Ruvi Tag settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ruuvitag.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ruuvitag]
platform = nordicnrf52
board = ruuvitag

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Ruuvi Tag supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:ruuvitag]
platform = nordicnrf52
board = ruuvitag

upload_protocol = jlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `<Tools & Debug Probes>` using `debug_tool` option in “`platformio.ini` (Project Configuration File)”.

Ruuvi Tag does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### SDT52832B

#### Contents

- **SDT52832B**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Sigma Delta Technologies</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `sdt52832b` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:sdt52832b]
platform = nordicnrf52
board = sdt52832b
```
You can override default SDT52832B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sdt52832b.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sdt52832b]
platform = nordicnrf52
board = sdt52832b

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

**Uploading**

SDT52832B supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:sdt52832b]
platform = nordicnrf52
board = sdt52832b

upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

SDT52832B does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | |
CMSIS-DAP | | |
J-LINK | | |
ST-LINK | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Taida Century nRF52 mini board

Contents

- Taida Century nRF52 mini board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Taida Century</td>
</tr>
</tbody>
</table>

Configuration

Please use stct_nrf52_minidev ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:stct_nrf52_minidev]
platform = nordinrf52
board = stct_nrf52_minidev
```
You can override default Taida Century nRF52 mini board settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `stct_nrf52_minidev.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:stct_nrf52_minidev]
platform = nordicnrf52
board = stct_nrf52_minidev

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

**Uploading**

Taida Century nRF52 mini board supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:stct_nrf52_minidev]
platform = nordicnrf52
board = stct_nrf52_minidev

upload_protocol = jlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Taida Century nRF52 mini board does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

#### Compatible Tools

- **Black Magic Probe**
- **CMSIS-DAP**
- **J-LINK**
- **ST-LINK**

### Frameworks

#### Name | Description
--- | ---
**Arduino** | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences

### VNG VBLUno52

#### Contents

- **VNG VBLUno52**
  - **Hardware**
  - **Configuration**
  - **Uploading**
  - **Debugging**
  - **Frameworks**

#### Hardware

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
<td>VNG</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `vbluno52` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:vbluno52]
platform = nordicnrf52
board = vbluno52
```
You can override default VNG VBLUno52 settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `vbluno52.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:vbluno52]
platform = nordicnrf52
board = vbluno52

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

VNG VBLUno52 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```ini
[env:vbluno52]
platform = nordicnrf52
board = vbluno52

upload_protocol = cmsis-dap
```

### Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

VNG VBLUno52 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.
## Compatible Tools

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### decaWave DWM1001 Module Development Board

#### Contents

- decaWave DWM1001 Module Development Board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>decaWave</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `dwm1001_dev` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:dwm1001_dev]
platform = nordicnrf52
board = dwm1001_dev
```
You can override default decaWave DWM1001 Module Development Board settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `dwm1001_dev.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:dwm1001_dev]
platform = nordicnrf52
board = dwm1001_dev

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

decaWave DWM1001 Module Development Board supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:dwm1001_dev]
platform = nordicnrf52
board = dwm1001_dev

upload_protocol = jlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

decaWave DWM1001 Module Development Board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### hackaBLE

#### Contents

- **hackaBLE**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52832</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Electronut Labs</td>
</tr>
</tbody>
</table>

#### Configuration

Please use hackaBLE ID for board option in “platformio.ini” (Project Configuration File):

```
[env:hackaBLE]
platform = nordicnrf52
board = hackaBLE
```
You can override default hackaBLE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hackaBLE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:hackaBLE]
platform = nordicnrf52
board = hackaBLE

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

hackaBLE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using `upload_protocol` option:

```
[env:hackaBLE]
platform = nordicnrf52
board = hackaBLE

upload_protocol = jlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

hackaBLE does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

reel_board

Contents

- reel_board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Nordic nRF52**: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PHYTEC</td>
</tr>
</tbody>
</table>

Configuration

Please use `reel_board` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:reel_board]
platform = nordicnrf52
board = reel_board
```

You can override default reel_board settings per build environment using `board_***` option, where *** is a JSON object path from board manifest reel_board.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:reel_board]
platform = nordicnrf52
board = reel_board
```

(continues on next page)
### Uploading

reel_board supports the next uploading protocols:
- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is cmsis-dap

You can change upload protocol using `upload_protocol` option:

```ini
[env:reel_board]
platform = nordicnrf52
board = reel_board
upload_protocol = cmsis-dap
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

reel_board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

reel_board_v2

Contents

- reel_board_v2
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>NRF52840</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PHYTEC</td>
</tr>
</tbody>
</table>

Configuration

Please use reel_board_v2 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:reel_board_v2]
platform = nordicnrf52
board = reel_board_v2
```

You can override default reel_board_v2 settings per build environment using board_*** option, where *** is a JSON object path from board manifest reel_board_v2.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:reel_board_v2]
platform = nordicnrf52
board = reel_board_v2
```

(continues on next page)
Uploading

reel_board_v2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```ini
[env:reel_board_v2]
platform = nordicnrf52
board = reel_board_v2
upload_protocol = cmsis-dap
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

reel_board_v2 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

u-blox EVK-NINA-B1

Contents

- u-blox EVK-NINA-B1
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nordic nRF52: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
<td>u-blox</td>
</tr>
</tbody>
</table>

Configuration

Please use ubox_evk_nina_b1 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ublox_evk_nina_b1]
platform = nordicnrf52
board = ublox_evk_nina_b1
```

You can override default u-blox EVK-NINA-B1 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ubox_evk_nina_b1.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ublox_evk_nina_b1]
platform = nordicnrf52
board = ublox_evk_nina_b1
```

(continues on next page)
Uploading

u-blox EVK-NINA-B1 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```plaintext
[env:ublox_evk_nina_b1]
platform = nordicnrf52
board = ublox_evk_nina_b1
upload_protocol = jlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in *platformio.ini* (Project Configuration File).

u-blox EVK-NINA-B1 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

1.12.21 Nuclei

GD32VF103V Evaluation Kit

Contents

- GD32VF103V Evaluation Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nuclei: Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GD32VF103VBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>108MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>GigaDevice</td>
</tr>
</tbody>
</table>

Configuration

Please use gd32vf103v_eval ID for board option in “platformio.ini” (Project Configuration File):

[env:gd32vf103v_eval]
platform = nuclei
board = gd32vf103v_eval

You can override default GD32VF103V Evaluation Kit settings per build environment using board_*** option, where *** is a JSON object path from board manifest gd32vf103v_eval.json. For example, board_build.mcu, board_build.f_cpu, etc.

1.12. Boards
[env:gd32vf103v_eval]
platform = nuclei
board = gd32vf103v_eval

; change microcontroller
board_build.mcu = gd32vf103vbt6

; change MCU frequency
board_build.f_cpu = 108000000L

Uploading

GD32VF103V Evaluation Kit supports the next uploading protocols:

- altera-usb-blaster
- gd-link
- jlink
- rv-link

Default protocol is rv-link

You can change upload protocol using upload_protocol option:

[env:gd32vf103v_eval]
platform = nuclei
board = gd32vf103v_eval
upload_protocol = rv-link

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

GD32VF103V Evaluation Kit does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera / Intel USB-Blaster</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>GD-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RV-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>

GD32VF103V RVStar Kit

Contents

- GD32VF103V RVStar Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nuclei: Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GD32VF103VBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>108MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nuclei</td>
</tr>
</tbody>
</table>

Configuration

Please use gd32vf103v_rvstar ID for board option in “platformio.ini” (Project Configuration File):

```
[env:gd32vf103v_rvstar]
platform = nuclei
board = gd32vf103v_rvstar
```

You can override default GD32VF103V RVStar Kit settings per build environment using board_*** option, where *** is a JSON object path from board manifest gd32vf103v_rvstar.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:gd32vf103v_rvstar]
platform = nuclei
board = gd32vf103v_rvstar

; change microcontroller
board_build.mcu = gd32vf103vbt6
```

(continues on next page)
}; change MCU frequency
board_build.f_cpu = 108000000L

Uploading

GD32VF103V RVStar Kit supports the next uploading protocols:

- altera-usb-blaster
- gd-link
- jlink
- rv-link

Default protocol is rv-link

You can change upload protocol using `upload_protocol` option:

```python
[env:gd32vf103v_rvstar]
platform = nuclei
board = gd32vf103v_rvstar
upload_protocol = rv-link
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

GD32VF103V RVStar Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera / Intel USB-Blaster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RV-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>
HummingBird Evaluation Kit

Contents

- HummingBird Evaluation Kit
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **Nuclei**: Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>HUMMINGBIRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>5MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Nuclei</td>
</tr>
</tbody>
</table>

Configuration

Please use `hbird_eval` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:hbird_eval]
platform = nuclei
board = hbird_eval
```

You can override default HummingBird Evaluation Kit settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `hbird_eval.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:hbird_eval]
platform = nuclei
board = hbird_eval

; change microcontroller
board_build.mcu = HummingBird

; change MCU frequency
board_build.f_cpu = 5000000L
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

HummingBird Evaluation Kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>

Sipeed Longan Nano

Contents

- Sipeed Longan Nano
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Nuclei: Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GD32VF103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>108MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>GigaDevice</td>
</tr>
</tbody>
</table>

Configuration

Please use `gd32vf103c_longan_nano` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default Sipeed Longan Nano settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `gd32vf103c_longan_nano.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:gd32vf103c_longan_nano]
platform = nuclei
board = gd32vf103c_longan_nano

; change microcontroller
board_build.mcu = gd32vf103cbt6

; change MCU frequency
board_build.f_cpu = 108000000L
```

**Uploading**

Sipeed Longan Nano supports the next uploading protocols:

- `altera-usb-blaster`
- `gd-link`
- `jlink`
- `rv-link`

Default protocol is `rv-link`

You can change upload protocol using `upload_protocol` option:

```ini
[env:gd32vf103c_longan_nano]
platform = nuclei
board = gd32vf103c_longan_nano

upload_protocol = rv-link
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Sipeed Longan Nano does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
---|---|---
Altera / Intel USB-Blaster | Yes | ---
GD-LINK | --- | ---
J-LINK | --- | ---
RV-LINK | --- | ---

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>

1.12.22 NXP i.MX RT

NXP i.MX RT1010 Evaluation Kit

Contents

- **NXP i.MX RT1010 Evaluation Kit**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP i.MX RT**: The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MIMXRT1011DAE5A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>500MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use `mimxrt1010_evk` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:mimxrt1010_evk]
platform = nxpimxrt
board = mimxrt1010_evk
```
You can override default NXP i.MX RT1010 Evaluation Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mimxrt1010_evk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mimxrt1010_evk]
platform = nxpimxrt
board = mimxrt1010_evk

; change microcontroller
board_build.mcu = mimxrt1011dae5a

; change MCU frequency
board_build.f_cpu = 500000000L
```

### Uploading

NXP i.MX RT1010 Evaluation Kit supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:mimxrt1010_evk]
platform = nxpimxrt
board = mimxrt1010_evk

upload_protocol = mbed
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

NXP i.MX RT1010 Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

NXP i.MX RT1015 Evaluation Kit

Contents

- NXP i.MX RT1015 Evaluation Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP i.MX RT: The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MIMXRT1015DAF5A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>500MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>96KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use mimxrt1015_evk ID for board option in "platformio.ini" (Project Configuration File):

```
[env:mimxrt1015_evk]
platform = nxpimxrt
board = mimxrt1015_evk
```

You can override default NXP i.MX RT1015 Evaluation Kit settings per build environment using board_*** option, where *** is a JSON object path from board manifest mimxrt1015_evk.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mimxrt1015_evk]
platform = nxpimxrt
board = mimxrt1015_evk
; change microcontroller
```

(continues on next page)
board_build.mcu = mimxrt1015daf5a

; change MCU frequency
board_build.f_cpu = 500000000L

Uploading

NXP i.MX RT1015 Evaluation Kit supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:mimxrt1015_evk]
platform = nxpimxrt
board = mimxrt1015_evk
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

NXP i.MX RT1015 Evaluation Kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>
NXP i.MX RT1020 Evaluation Kit

Contents

- NXP i.MX RT1020 Evaluation Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP i.MX RT: The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MIMXRT1021DAG5A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>500MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use mimxrt1020_evk ID for board option in "platformio.ini" (Project Configuration File):

```
[env:mimxrt1020_evk]
platform = nxpimxrt
board = mimxrt1020_evk
```

You can override default NXP i.MX RT1020 Evaluation Kit settings per build environment using board_*** option, where *** is a JSON object path from board manifest mimxrt1020_evk.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mimxrt1020_evk]
platform = nxpimxrt
board = mimxrt1020_evk

; change microcontroller
board_build.mcu = mimxrt1021dag5a

; change MCU frequency
board_build.f_cpu = 500000000L
```

Uploading

NXP i.MX RT1020 Evaluation Kit supports the next uploading protocols:
• blackmagic
• jlink
• mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:mimxrt1020_evk]
platform = nxpimxrt
board = mimxrt1020_evk
upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

NXP i.MX RT1020 Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

**NXP i.MX RT1050 Evaluation Kit**

**Contents**

• NXP i.MX RT1050 Evaluation Kit
  – Hardware
  – Configuration
  – Uploading
Hardware

Platform **NXP i.MX RT**: The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MIMXRT1052DVL6B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>600MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use `mimxrt1050_evk` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:mimxrt1050_evk]
platform = nxpimxrt
board = mimxrt1050_evk
```

You can override default NXP i.MX RT1050 Evaluation Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `mimxrt1050_evk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mimxrt1050_evk]
platform = nxpimxrt
board = mimxrt1050_evk

; change microcontroller
board_build.mcu = mimxrt1052dvl6b

; change MCU frequency
board_build.f_cpu = 600000000L
```

Uploading

NXP i.MX RT1050 Evaluation Kit supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

NXP i.MX RT1050 Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephy RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

NXP i.MX RT1060 Evaluation Kit

**Contents**

- **NXP i.MX RT1060 Evaluation Kit**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *NXP i.MX RT*: The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MIMXRT1062DVL6A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>600MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use `mimxrt1060_evk` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:mimxrt1060_evk]
platform = nxpimxrt
board = mimxrt1060_evk
```

You can override default NXP i.MX RT1060 Evaluation Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `mimxrt1060_evk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mimxrt1060_evk]
platform = nxpimxrt
board = mimxrt1060_evk

; change microcontroller
board_build.mcu = mimxrt1062dvl6a

; change MCU frequency
board_build.f_cpu = 600000000L
```

Uploading

NXP i.MX RT1060 Evaluation Kit supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:mimxrt1060_evk]
platform = nxpimxrt
board = mimxrt1060_evk

upload_protocol = mbed
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

NXP i.MX RT1060 Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

NXP i.MX RT1064 Evaluation Kit

Contents

- **NXP i.MX RT1064 Evaluation Kit**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP i.MX RT**: The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MIMXRT1064DVL6A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>600MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>
Configuration

Please use mimxrt1064_evk ID for board option in "platformio.ini" (Project Configuration File):

```
[env:mimxrt1064_evk]
platform = nxpimxrt
board = mimxrt1064_evk
```

You can override default NXP i.MX RT1064 Evaluation Kit settings per build environment using board_*** option, where *** is a JSON object path from board manifest mimxrt1064_evk.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mimxrt1064_evk]
platform = nxpimxrt
board = mimxrt1064_evk

; change microcontroller
board_build.mcu = mimxrt1064dvl6a

; change MCU frequency
board_build.f_cpu = 600000000L
```

Uploading

NXP i.MX RT1064 Evaluation Kit supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:mimxrt1064_evk]
platform = nxpimxrt
board = mimxrt1064_evk

upload_protocol = mbed
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

NXP i.MX RT1064 Evaluation Kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe |  | 
J-LINK | Yes | Yes

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

1.12.23 NXP LPC

ARM mbed LPC11U24 (+CAN)

Contents

- ARM mbed LPC11U24 (+CAN)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc11u24_301 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:lpc11u24_301]
platform = nxplpc
board = lpc11u24_301
```
You can override default ARM mbed LPC11U24 (+CAN) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u24_301.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lpc11u24_301]
platform = nxplpc
board = lpc11u24_301

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

ARM mbed LPC11U24 (+CAN) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:lpc11u24_301]
platform = nxplpc
board = lpc11u24_301

upload_protocol = mbed
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in **“platformio.ini”** (Project Configuration File).

ARM mbed LPC11U24 (+CAN) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Bambino-210E

Contents

- Bambino-210E
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC4330</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>204MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8MB</td>
</tr>
<tr>
<td>RAM</td>
<td>264KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Micromint</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc4330_m4 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lpc4330_m4]
platform = nxplpc
board = lpc4330_m4
```

You can override default Bambino-210E settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpc4330_m4.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

Bambino-210E supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:lpc4330_m4]
platform = nxp1pc
board = lpc4330_m4

upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

Bambino-210E has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

CQ Publishing TG-LPC11U35-501

Contents

- CQ Publishing TG-LPC11U35-501
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>CQ Publishing</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpc11u35_501` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:lpc11u35_501]
platform = nxplpc
board = lpc11u35_501
```

You can override default CQ Publishing TG-LPC11U35-501 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u35_501.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:lpcl1u35_501]
platform = nxplpc
board = lpcl1u35_501

; change microcontroller
board_build.mcu = lpcl1u35

; change MCU frequency
board_build.f_cpu = 48000000L

### Uploading

CQ Publishing TG-LPC11U35-501 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

[env:lpcl1u35_501]
platform = nxplpc
board = lpcl1u35_501
upload_protocol = mbed

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

CQ Publishing TG-LPC11U35-501 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

CoCo-ri-Co!

Contents

- CoCo-ri-Co!
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC812</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>30MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Elektor Labs</td>
</tr>
</tbody>
</table>

Configuration

Please use `elektor_cocorico` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:elektor_cocorico]
platform = nxplpc
board = elektor_cocorico
```

You can override default CoCo-ri-Co! settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `elektor_cocorico.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

CoCo-ri-Co! supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:elektor_cocorico]
platform = nxplpc
board = elektor_cocorico
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

CoCo-ri-Co! has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

DipCortex M3

Contents

- DipCortex M3
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC1347</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>12KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Solder Splash Labs</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc1347 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lpc1347]
platform = nxplpc
board = lpc1347
```

You can override default DipCortex M3 settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpc1347.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

DipCortex M3 supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:lpc1347]
platform = nxplpc
board = lpc1347

upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (*Project Configuration File)*.

DipCortex M3 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

**Mbed**

Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.
EA LPC11U35 QuickStart Board

Contents

• EA LPC11U35 QuickStart Board
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Embedded Artists</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc11u35 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lpc11u35]
platform = nxplpc
board = lpc11u35
```

You can override default EA LPC11U35 QuickStart Board settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpc11u35.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:lpc11u35]
platform = nxplpc
board = lpc11u35

; change microcontroller
board_build.mcu = lpc11u35

; change MCU frequency
board_build.f_cpu = 48000000L
```
Uploading

EA LPC11U35 QuickStart Board supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:lpc11u35]
platform = nxplpc
board = lpc11u35
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

EA LPC11U35 QuickStart Board does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Embedded Artists LPC4088 Display Module

Contents

- Embedded Artists LPC4088 Display Module
Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC4088</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Embedded Artists</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpc4088_dm` ID for `board` option in “`platformio.ini` (Project Configuration File)”: 

```
[env:lpc4088_dm]
platform = nxplpc
board = lpc4088_dm
```

You can override default Embedded Artists LPC4088 Display Module settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc4088_dm.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc4088_dm]
platform = nxplpc
board = lpc4088_dm

; change microcontroller
board_build.mcu = lpc4088

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Embedded Artists LPC4088 Display Module supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed
Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:lpc4088_dm]
platform = nxplpc
board = lpc4088_dm

upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Embedded Artists LPC4088 Display Module has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**Embedded Artists LPC4088 QuickStart Board**

**Contents**

- *Embedded Artists LPC4088 QuickStart Board*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC4088</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Embedded Artists</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc4088 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lpc4088]
platform = nxplpc
board = lpc4088
```

You can override default Embedded Artists LPC4088 QuickStart Board settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpc4088.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:lpc4088]
platform = nxplpc
board = lpc4088

; change microcontroller
board_build.mcu = lpc4088

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Embedded Artists LPC4088 QuickStart Board supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:lpc4088]
platform = nxplpc
board = lpc4088

upload_protocol = mbed
```
Debugging

"1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Embedded Artists LPC4088 QuickStart Board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**LPCXpresso11U68**

**Contents**

- LPCXpresso11U68
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *NXP LPC*: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.
**Configuration**

Please use `lpc11u68` ID for `board` option in `platformio.ini` (Project Configuration File):

```
[env:lpc11u68]
platform = nxplpc
board = lpc11u68
```

You can override default LPCXpresso11U68 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u68.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc11u68]
platform = nxplpc
board = lpc11u68

; change microcontroller
board_build.mcu = lpc11u68

; change MCU frequency
board_build.f_cpu = 50000000L
```

**Uploading**

LPCXpresso11U68 supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `jlink`
- `mbed`

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:lpc11u68]
platform = nxplpc
board = lpc11u68

upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

LPCXpresso11U68 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

LPCXpresso824-MAX

Contents

- LPCXpresso824-MAX
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC824</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>30MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

### Configuration

Please use `lpc824` ID for `board` option in “platformio.ini” (*Project Configuration File)*:

```ini
[env:lpc824]
platform = nxplpc
board = lpc824
```

You can override default LPCXpresso824-MAX settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc824.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.:

```ini
[env:lpc824]
platform = nxplpc
board = lpc824

; change microcontroller
board_build.mcu = lpc824

; change MCU frequency
board_build.f_cpu = 30000000L
```

### Uploading

LPCXpresso824-MAX supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:lpc824]
platform = nxplpc
board = lpc824

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

LPCXpresso824-MAX has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

MicroNFCBoard

Contents

- MicroNFCBoard
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U34</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>48KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>AppNearMe</td>
</tr>
</tbody>
</table>
Configuration

Please use micronfcboard ID for board option in "platformio.ini" (Project Configuration File):

```
[env:micronfcboard]
platform = nxplpc
board = micronfcboard
```

You can override default MicroNFCBoard settings per build environment using board_*** option, where *** is a JSON object path from board manifest micronfcboard.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:micronfcboard]
platform = nxplpc
board = micronfcboard

; change microcontroller
board_build.mcu = lpc11u34

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

Debugging currently does not support MicroNFCBoard board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

NGX Technologies BlueBoard-LPC11U24

Contents

- NGX Technologies BlueBoard-LPC11U24
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NGX Technologies</td>
</tr>
</tbody>
</table>

Configuration

Please use `blueboard_lpc11u24` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:blueboard_lpc11u24]
platform = nxplpc
board = blueboard_lpc11u24
```

You can override default NGX Technologies BlueBoard-LPC11U24 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `blueboard_lpc11u24.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:blueboard_lpc11u24]
platform = nxplpc
board = blueboard_lpc11u24

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

NGX Technologies BlueBoard-LPC11U24 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:blueboard_lpc11u24]
platform = nxplpc
board = blueboard_lpc11u24

upload_protocol = mbed
```
Debugging

"1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" *(Project Configuration File)*.

NGX Technologies BlueBoard-LPC11U24 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

NXP LPC11C24

**Contents**

- *NXP LPC11C24*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *NXP LPC*: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11C24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

**Configuration**

Please use lpc11c24 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:lpc11c24]
platform = nxplpc
board = lpc11c24
```

You can override default NXP LPC11C24 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest lpc11c24.json. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:lpc11c24]
platform = nxplpc
board = lpc11c24

; change microcontroller
board_build.mcu = lpc11c24

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

NXP LPC11C24 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:lpc11c24]
platform = nxplpc
board = lpc11c24

upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

NXP LPC11C24 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

NXP LPC11U34

Contents

- NXP LPC11U34
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U34</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>40KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>
Configuration

Please use lpc11u34_421 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lpc11u34_421]
platform = nxplpc
board = lpc11u34_421
```

You can override default NXP LPC11U34 settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpc11u34_421.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:lpc11u34_421]
platform = nxplpc
board = lpc11u34_421

; change microcontroller
board_build.mcu = lpc11u34

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

NXP LPC11U34 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:lpc11u34_421]
platform = nxplpc
board = lpc11u34_421

upload_protocol = mbed
```

Debugging

*Debugging*- “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

NXP LPC11U34 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### NXP LPC11U37

#### Contents

- **NXP LPC11U37**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `lpc11u37_501` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:lpc11u37_501]
platform = nxplpc
board = lpc11u37_501
```

---

1.12. Boards
You can override default NXP LPC11U37 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lpc11u37_501.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc11u37_501]
platform = nxplpc
board = lpc11u37_501

; change microcontroller
board_build.mcu = lpc11u37

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

NXP LPC11U37 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:lpc11u37_501]
platform = nxplpc
board = lpc11u37_501

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

NXP LPC11U37 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
</tbody>
</table>

NXP LPC800-MAX

Contents

- NXP LPC800-MAX
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC812</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>30MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc812 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:lpc812]
platform = nxplpc
board = lpc812
```

You can override default NXP LPC800-MAX settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpc812.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env: lpc812]
platform = nxplpc
board = lpc812

; change microcontroller
board_build.mcu = lpc812

; change MCU frequency
board_build.f_cpu = 30000000L

### Uploading

NXP LPC800-MAX supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

[env: lpc812]
platform = nxplpc
board = lpc812

upload_protocol = mbed

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

NXP LPC800-MAX has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

NXP LPCXpresso1549

Contents

- **NXP LPCXpresso1549**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC1549</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>36KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpc1549` ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lpc1549]
platform = nxplpc
board = lpc1549
```

You can override default NXP LPCXpresso1549 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lpc1549.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.
[env: lpc1549]
platform = nxplpc
board = lpc1549

; change microcontroller
board_build.mcu = lpc1549

; change MCU frequency
board_build.f_cpu = 72000000L

**Uploading**

NXP LPCXpresso1549 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

[env: lpc1549]
platform = nxplpc
board = lpc1549

upload_protocol = mbed

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

NXP LPCXpresso1549 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

NXP LPCXpresso54114

Contents

- NXP LPCXpresso54114
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC54114J256BD64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpc54114` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:lpc54114]
platform = nxplpc
board = lpc54114
```

You can override default NXP LPCXpresso54114 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc54114.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

NXP LPCXpresso54114 supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:lpc54114]
platform = nxplpc
board = lpc54114
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

NXP LPCXpresso54114 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

NXP LPCXpresso54608

Contents

- NXP LPCXpresso54608
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC54608ET512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>200KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc546xx ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:lpc546xx]
platform = nxplpc
board = lpc546xx
```

You can override default NXP LPCXpresso54608 settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpc546xx.json. For example, board_build.mcu, board_build.f_cpu, etc.
### Uploading

NXP LPCXpresso54608 supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:lpc546xx]
platform = nxplpc
board = lpc546xx

upload_protocol = mbed
```

### Debugging

_Autoformatting_

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging _Tools & Debug Probes_ using `debug_tool` option in “platformio.ini” (Project Configuration File).

NXP LPCXpresso54608 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>J-LINK</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>
NXP LPCXpresso55S16

Contents

- NXP LPCXpresso55S16
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC55S16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>150MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use lpcxpresso55s16 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:lpcxpresso55s16]
platform = nxplpc
board = lpcxpresso55s16
```

You can override default NXP LPCXpresso55S16 settings per build environment using board_*** option, where *** is a JSON object path from board manifest lpcxpresso55s16.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:lpcxpresso55s16]
platform = nxplpc
board = lpcxpresso55s16

; change microcontroller
board_build.mcu = lpc55s16

; change MCU frequency
board_build.f_cpu = 150000000L
```
Uploading

NXP LPCXpresso55S16 supports the next uploading protocols:

- jlink
- mbed

Default protocol is jlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:lpcxpresso55s16]
platform = nxplpc
board = lpcxpresso55s16
upload_protocol = jlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

NXP LPCXpresso55S16 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

NXP LPCXpresso55S69

Contents

- **NXP LPCXpresso55S69**
  - Hardware
  - Configuration
Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC55S69</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>150MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>640KB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpcxpresso55s69` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:lpcxpresso55s69]
platform = nxplpc
board = lpcxpresso55s69
```

You can override default NXP LPCXpresso55S69 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpcxpresso55s69.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lpcxpresso55s69]
platform = nxplpc
board = lpcxpresso55s69

; change microcontroller
board_build.mcu = lpc55s69

; change MCU frequency
board_build.f_cpu = 150000000L
```

Uploading

NXP LPCXpresso55S69 supports the next uploading protocols:

- jlink
- mbed

Default protocol is jlink

You can change upload protocol using `upload_protocol` option:
### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

NXP LPCXpresso55S69 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### NXP mbed LPC11U24

**Contents**

- *NXP mbed LPC11U24*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpc11u24` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:lpc11u24]
platform = nxplpc
board = lpc11u24
```

You can override default NXP mbed LPC11U24 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u24.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc11u24]
platform = nxplpc
board = lpc11u24

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

NXP mbed LPC11U24 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:lpc11u24]
platform = nxplpc
board = lpc11u24
```

(continues on next page)
upload_protocol = mbed

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

NXP mbed LPC11U24 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

NXP mbed LPC1768

Contents

- **NXP mbed LPC1768**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC1768</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>NXP</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc1768 ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:lpc1768]
platform = nxplpc
board = lpc1768
```

You can override default NXP mbed LPC1768 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lpc1768.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lpc1768]
platform = nxplpc
board = lpc1768

; change microcontroller
board_build.mcu = lpc1768

; change MCU frequency
board_build.f_cpu = 96000000L
```

Uploading

NXP mbed LPC1768 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:lpc1768]
platform = nxplpc
board = lpc1768
```

(continues on next page)
upload_protocol = mbed

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

NXP mbed LPC1768 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**Outrageous Circuits mBuino**

**Contents**

- *Outrageous Circuits mBuino*
  - **Hardware**
  - **Configuration**
  - **Debugging**
  - **Frameworks**

**Hardware**

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the
Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Outrageous Circuits</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `mbuino` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:mbuino]
platform = nxplpc
board = mbuino
```

You can override default Outrageous Circuits mBuino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mbuino.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:mbuino]
platform = nxplpc
board = mbuino

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Debugging**

*Debugging* currently does not support Outrageous Circuits mBuino board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**Seeed Arch GPRS V2**

**Contents**

- Seeed Arch GPRS V2
Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>

Configuration

Please use `seeedArchGPRS` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:seeedArchGPRS]
platform = nxplpc
board = seeedArchGPRS
```

You can override default Seeed Arch GPRS V2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `seeedArchGPRS.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:seeedArchGPRS]
platform = nxplpc
board = seeedArchGPRS

; change microcontroller
board_build.mcu = lpc11u37

; change MCU frequency
board_build.f_cpu = 48000000
```

Debugging

*Debugging* currently does not support Seeed Arch GPRS V2 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Seeed Arch Pro

Contents

- Seeed Arch Pro
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *NXP LPC*: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC1768</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>

Configuration

Please use `seeedArchPro` ID for `board` option in “platformio.ini” (Project Configuration File): 

```
[env:seeedArchPro]
platform = nxplpc
board = seeedArchPro
```

You can override default Seeed Arch Pro settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `seeedArchPro.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.
Uploading

Seeed Arch Pro supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:seeedArchPro]
platform = nxplpc
board = seeedArchPro
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Seeed Arch Pro has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>
Seeed Xadow M0

Contents

- Seeed Xadow M0
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>

Configuration

Please use `xadow_m0` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:xadow_m0]
platform = nxplpc
board = xadow_m0
```

You can override default Seeed Xadow M0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xadow_m0.json`. For example, `board_build.mcu,board_build.f_cpu, etc.`

```ini
[env:xadow_m0]
platform = nxplpc
board = xadow_m0

; change microcontroller
board_build.mcu = lpc11u35

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

*Debugging* currently does not support Seeed Xadow M0 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Smeshlink xbed LPC1768

Contents

- Smeshlink xbed LPC1768
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC1768</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Smeshlink</td>
</tr>
</tbody>
</table>

Configuration

Please use `xbed_lpc1768` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:xbed_lpc1768]
platform = nxplpc
board = xbed_lpc1768
```

You can override default Smeshlink xbed LPC1768 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `xbed_lpc1768.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:xbed_lpc1768]
platform = nxplpc
board = xbed_lpc1768
```

(continues on next page)
; change microcontroller
board_build.mcu = lpc1768

; change MCU frequency
board_build.f_cpu = 96000000L

### Debugging

*Debugging* currently does not support Smeshlink xbed LPC1768 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Solder Splash Labs DipCortex M0

#### Contents

- *Solder Splash Labs DipCortex M0*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Solder Splash Labs</td>
</tr>
</tbody>
</table>
Configuration

Please use `dipcortexm0` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:dipcortexm0]
platform = nxplpc
board = dipcortexm0
```

You can override default Solder Splash Labs DipCortex M0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `dipcortexm0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:dipcortexm0]
platform = nxplpc
board = dipcortexm0

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Solder Splash Labs DipCortex M0 supports the next uploading protocols:

- `blackmagic`
- `jlink`
- `mbed`

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:dipcortexm0]
platform = nxplpc
board = dipcortexm0

upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “`platformio.ini` (Project Configuration File).

Solder Splash Labs DipCortex M0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Switch Science mbed LPC1114FN28

Contents

- Switch Science mbed LPC1114FN28
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC1114FN28</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Switch Science</td>
</tr>
</tbody>
</table>

Configuration

Please use lpc1114fn28 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lpc1114fn28]
platform = nxplpc
board = lpc1114fn28
```
You can override default Switch Science mbed LPC1114FN28 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lpc1114fn28.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:lpc1114fn28]
platform = nxplpc
board = lpc1114fn28

; change microcontroller
board_build.mcu = lpc1114fn28

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Switch Science mbed LPC1114FN28 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed.

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:lpc1114fn28]
platform = nxplpc
board = lpc1114fn28

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Switch Science mbed LPC1114FN28 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Switch Science mbed LPC824

Contents

- Switch Science mbed LPC824
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC824</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>30MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Switch Science</td>
</tr>
</tbody>
</table>

Configuration

Please use scci824 ID for board option in "platformio.ini" *(Project Configuration File)*:

```
[env:scci824]
platform = nxplpc
board = scci824
```

You can override default Switch Science mbed LPC824 settings per build environment using board_*** option, where *** is a JSON object path from board manifest scci824.json. For example, board_build.mcu, board_build.f_cpu, etc.
Uploading

Switch Science mbed LPC824 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:ssci824]
platform = nxplpc
board = ssci824

upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Switch Science mbed LPC824 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CMSIS-DAP</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>J-LINK</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

mBuino

Contents

- mBuino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U24</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>GHI Electronics</td>
</tr>
</tbody>
</table>

Configuration

Please use `oc_mbuino` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:oc_mbuino]
platform = nxplpc
board = oc_mbuino
```

You can override default mBuino settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `oc_mbuino.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```ini
[env:oc_mbuino]
platform = nxplpc
board = oc_mbuino

; change microcontroller
```

(continues on next page)
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 50000000L

Debugging

Debugging currently does not support mBuino board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

u-blox C027

Contents

- u-blox C027
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform NXP LPC: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC1768</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>96MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>u-blox</td>
</tr>
</tbody>
</table>
**Configuration**

Please use `ubloxc027` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:ubloxc027]
platform = nxplpc
board = ubloxc027
```

You can override default u-blox C027 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ubloxc027.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ubloxc027]
platform = nxplpc
board = ubloxc027

; change microcontroller
board_build.mcu = lpc1768

; change MCU frequency
board_build.f_cpu = 96000000L
```

**Uploading**

u-blox C027 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:ubloxc027]
platform = nxplpc
board = ubloxc027

upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

u-blox C027 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

**Name** | **Description**
---|---
Mbed | Arm Mbed OS is an open source embedded operating system designed specifically for the 'things' in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.

### y5 LPC11U35 mbug

#### Contents

- y5 LPC11U35 mbug
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **NXP LPC**: The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>y5 design</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `lpc11u35_y5_mbug` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:lpc11u35_y5_mbug]
platform = nxplpc
board = lpc11u35_y5_mbug
```
You can override default y5 LPC11U35 mbug settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lpc11u35_y5_mbug.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lpc11u35_y5_mbug]
platform = nxplpc
board = lpc11u35_y5_mbug

; change microcontroller
board_build.mcu = lpc11u35

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

y5 LPC11U35 mbug supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:lpc11u35_y5_mbug]
platform = nxplpc
board = lpc11u35_y5_mbug

upload_protocol = mbed
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

y5 LPC11U35 mbug does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

1.12.24 RISC-V GAP

GAPuino GAP8

Contents

- GAPuino GAP8
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform RISC-V GAP: GreenWaves GAP8 IoT application processor enables the cost-effective development, deployment and autonomous operation of intelligent sensing devices that capture, analyze, classify and act on the fusion of rich data sources such as images, sounds or vibrations.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>GAP8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>250MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64MB</td>
</tr>
<tr>
<td>RAM</td>
<td>8MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>GreenWaves Technologies</td>
</tr>
</tbody>
</table>

Configuration

Please use gapuino ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:gapuino]
platform = riscv_gap
board = gapuino
```

You can override default GAPuino GAP8 settings per build environment using board_*** option, where *** is a JSON object path from board manifest gapuino.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:gapuino]
platform = riscv_gap
board = gapuino

; change microcontroller
board_build.mcu = gap8

; change MCU frequency
board_build.f_cpu = 250000000L

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

GAPuino GAP8 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>PULP OS</td>
<td>PULP is a silicon-proven Parallel Ultra Low Power platform targeting high energy efficiencies. The platform is organized in clusters of RISC-V cores that share a tightly-coupled data memory</td>
</tr>
</tbody>
</table>

### 1.12.25 Shakti

**Artix-7 35T Arty FPGA Evaluation Kit**

**Contents**

- *Artix-7 35T Arty FPGA Evaluation Kit*
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform **Shakti**: Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

```
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>E-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>0B</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Xilinx</td>
</tr>
</tbody>
</table>
```

Configuration

Please use `artix7_35t` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:artix7_35t]
platform = shakti
board = artix7_35t
```

You can override default Artix-7 35T Arty FPGA Evaluation Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `artix7_35t.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:artix7_35t]
platform = shakti
board = artix7_35t

; change microcontroller
board_build.mcu = E-Class

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Artix-7 35T Arty FPGA Evaluation Kit supports the next uploading protocols:

- ftdi
- ftdi
- jlink
- jlink

Default protocol is ftdi.

You can change upload protocol using `upload_protocol` option:
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" *(Project Configuration File)*.

Artix-7 35T Arty FPGA Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakti SDK</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
</tbody>
</table>

**Arty A7-100: Artix-7 FPGA Development Board**

**Contents**

- Arty A7-100: Artix-7 FPGA Development Board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Shakti*: Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools,
storage controllers, peripheral IPs and SOC tools.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>C-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>0B</td>
</tr>
<tr>
<td>RAM</td>
<td>128MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Xilinx</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `artix7_100t` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:artix7_100t]
platform = shakti
board = artix7_100t
```

You can override default Arty A7-100: Artix-7 FPGA Development Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `artix7_100t.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:artix7_100t]
platform = shakti
board = artix7_100t

; change microcontroller
board_build.mcu = C-Class

; change MCU frequency
board_build.f_cpu = 50000000L
```

**Uploading**

Arty A7-100: Artix-7 FPGA Development Board supports the next uploading protocols:

- ftdi
- jlink
- jlink

Default protocol is ftdi

You can change upload protocol using `upload_protocol` option:

```ini
[env:artix7_100t]
platform = shakti
board = artix7_100t

upload_protocol = ftdi
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in "`platformio.ini`" (*Project Configuration File)*.

Arty A7-100: Artix-7 FPGA Development Board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Shakti SDK</em></td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
</tbody>
</table>

**Parashu on Artix-7 100T Arty FPGA Evaluation Kit**

**Contents**

- *Parashu on Artix-7 100T Arty FPGA Evaluation Kit*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Shakti*: Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>E-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>0B</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Xilinx</td>
</tr>
</tbody>
</table>
Configuration

Please use `parashu` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:parashu]
platform = shakti
board = parashu
```

You can override default Parashu on Artix-7 100T Arty FPGA Evaluation Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `parashu.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:parashu]
platform = shakti
board = parashu

; change microcontroller
board_build.mcu = E-Class

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Parashu on Artix-7 100T Arty FPGA Evaluation Kit supports the next uploading protocols:

- ftdi
- ftdi
- jlink
- jlink

Default protocol is `ftdi`

You can change upload protocol using `upload_protocol` option:

```ini
[env:parashu]
platform = shakti
board = parashu

upload_protocol = ftdi
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
Parashu on Artix-7 100T Arty FPGA Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakti SDK</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
</tbody>
</table>

### Pinaka on Artix-7 35T Arty FPGA Evaluation Kit

#### Contents

- *Pinaka on Artix-7 35T Arty FPGA Evaluation Kit*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *Shakti*: Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>E-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>0B</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Xilinx</td>
</tr>
</tbody>
</table>

### Configuration

Please use `pinaka` ID for `board` option in "platformio.ini" *(Project Configuration File):*

```
[env:pinaka]
platform = shakti
board = pinaka
```
You can override default Pinaka on Artix-7 35T Arty FPGA Evaluation Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest pinaka.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pinaka]
platform = shakti
board = pinaka

; change microcontroller
board_build.mcu = E-Class

; change MCU frequency
board_build.f_cpu = 50000000L
```

## Uploading

Pinaka on Artix-7 35T Arty FPGA Evaluation Kit supports the next uploading protocols:

- ftdi
- ftdi
- jlink
- jlink

Default protocol is ftdi.

You can change upload protocol using `upload_protocol` option:

```
[env:pinaka]
platform = shakti
board = pinaka

upload_protocol = ftdi
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

Pinaka on Artix-7 35T Arty FPGA Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakti SDK</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
</tbody>
</table>

Vajra on Arty A7-100: Artix-7 FPGA Development Board

Contents

- Vajra on Arty A7-100: Artix-7 FPGA Development Board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **Shakti**: Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>C-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>0B</td>
</tr>
<tr>
<td>RAM</td>
<td>128MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Xilinx</td>
</tr>
</tbody>
</table>

Configuration

Please use `vajra` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:vajra]
platform = shakti
board = vajra
```

You can override default Vajra on Arty A7-100: Artix-7 FPGA Development Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `vajra.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:vajra]
platform = shakti
board = vajra

; change microcontroller
```

(continues on next page)
Uploading

Vajra on Arty A7-100: Artix-7 FPGA Development Board supports the next uploading protocols:

- ftdi
- ftdi
- jlink
- jlink

Default protocol is ftdi

You can change upload protocol using upload_protocol option:

```ini
[env:vajra]
platform = shakti
board = vajra
upload_protocol = ftdi
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

Vajra on Arty A7-100: Artix-7 FPGA Development Board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakti SDK</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
</tbody>
</table>
1.12.26 SiFive

Arty FPGA Dev Kit

Contents

- Arty FPGA Dev Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **SiFive**: SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>FE310</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>450MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Xilinx</td>
</tr>
</tbody>
</table>

Configuration

Please use `e310-arty` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:e310-arty]
platform = sifive
board = e310-arty
```

You can override default Arty FPGA Dev Kit settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `e310-arty.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:e310-arty]
platform = sifive
board = e310-arty

; change microcontroller
board_build.mcu = fe310

; change MCU frequency
board_build.f_cpu = 450000000L
```
Uploading

Arty FPGA Dev Kit supports the next uploading protocols:

- ftdi
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is ftdi

You can change upload protocol using `upload_protocol` option:

```
[env:e310-arty]
platform = sifive
board = e310-arty
upload_protocol = ftdi
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Arty FPGA Dev Kit has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mini-Module FT2232H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCD-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Renode</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>TIAO USB Multi-Protocol Adapter (TUMPA)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
</tbody>
</table>

HiFive Unleashed

Contents

- HiFive Unleashed
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **SiFive**: SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>FU540</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1500MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32MB</td>
</tr>
<tr>
<td>RAM</td>
<td>8GB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SiFive</td>
</tr>
</tbody>
</table>

Configuration

Please use `hifive-unleashed` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:hifive-unleashed]
platform = sifive
board = hifive-unleashed
```

You can override default HiFive Unleashed settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `hifive-unleashed.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:hifive-unleashed]
platform = sifive
board = hifive-unleashed

; change microcontroller
```

(continues on next page)
board_build.mcu = fu540
; change MCU frequency
board_build.f_cpu = 1500000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

HiFive Unleashed has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>QEMU</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Renode</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
</tbody>
</table>

HiFive1

Contents

- HiFive1
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **SiFive**: SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>FE310</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>320MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SiFive</td>
</tr>
</tbody>
</table>

Configuration

Please use `hifive1` ID for `board` option in “`platformio.ini` (Project Configuration File)“:

```
[env:hifive1]
platform = sifive
board = hifive
```

You can override default HiFive1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hifive1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:hifive1]
platform = sifive
board = hifive

; change microcontroller
board_build.mcu = fe310

; change MCU frequency
board_build.f_cpu = 320000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File)“.

HiFive1 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTDI Chip</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>QEMU</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Renode</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

HiFive1 Rev B

Contents

- HiFive1 Rev B
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform SiFive: SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
<td>SiFive</td>
</tr>
</tbody>
</table>

Configuration

Please use `hifive1-revb` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:hifive1-revb]
platform = sifive
board = hifive1-revb
```

You can override default HiFive1 Rev B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hifive1-revb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:hifivel-revb]
platform = sifive
board = hifivel-revb

; change microcontroller
board_build.mcu = fe310

; change MCU frequency
board_build.f_cpu = 320000000L

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

HiFive1 Rev B has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Renode</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### SparkFun RED-V RedBoard

#### Contents

- SparkFun RED-V RedBoard
  - Hardware
  - Configuration
PlatformIO Documentation, Release 5.0.5a1

- Debugging
- Frameworks

Hardware

Platform SiFive: SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>FE310</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>320MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

Configuration

Please use sparkfun_redboard_v ID for board option in “platformio.ini” (Project Configuration File):

```
[env:sparkfun_redboard_v]
platform = sifive
board = sparkfun_redboard_v
```

You can override default SparkFun RED-V RedBoard settings per build environment using board_*** option, where *** is a JSON object path from board manifest sparkfun_redboard_v.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:sparkfun_redboard_v]
platform = sifive
board = sparkfun_redboard_v

; change microcontroller
board_build.mcu = fe310

; change MCU frequency
board_build.f_cpu = 320000000L
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

SparkFun RED-V RedBoard has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Renode</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### SparkFun RED-V Thing Plus

**Contents**

- SparkFun RED-V Thing Plus
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **SiFive**: SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>FE310</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>320MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SparkFun</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `sparkfun_thing_plus_v` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:sparkfun_thing_plus_v]
platform = sifive
board = sparkfun_thing_plus_v
```
You can override default SparkFun RED-V Thing Plus settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_thing_plus_v.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:sparkfun_thing_plus_v]
platform = sifive
board = sparkfun_thing_plus_v

; change microcontroller
board_build.mcu = fe310

; change MCU frequency
board_build.f_cpu = 320000000L
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “`platformio.ini` (Project Configuration File).

SparkFun RED-V Thing Plus has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Renode</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### 1.12.27 Silicon Labs EFM32

**EFM32GG-STK3700 Giant Gecko**
**Hardware**

Platform *Silicon Labs EFM32*: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFM32GG990F1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `efm32gg_stk3700` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:efm32gg_stk3700]
platform = siliconlabsefm32
board = efm32gg_stk3700
```

You can override default EFM32GG-STK3700 Giant Gecko settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `efm32gg_stk3700.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:efm32gg_stk3700]
platform = siliconlabsefm32
board = efm32gg_stk3700

; change microcontroller
board_build.mcu = efm32gg990f1024

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

EFM32GG-STK3700 Giant Gecko supports the next uploading protocols:
Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:efm32gg_stk3700]
platform = siliconlabsefm32
board = efm32gg_stk3700
upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

EFM32GG-STK3700 Giant Gecko has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

**Mbed**

Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.

**EFM32LG-STK3600 Leopard Gecko**

**Contents**

- EFM32LG-STK3600 Leopard Gecko
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform **Silicon Labs EFM32**: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFM32LG990F256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>

Configuration

Please use `efm32lg_stk3600` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```
[env:efm32lg_stk3600]
platform = siliconlabsefm32
board = efm32lg_stk3600
```

You can override default EFM32LG-STK3600 Leopard Gecko settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `efm32lg_stk3600.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:efm32lg_stk3600]
platform = siliconlabsefm32
board = efm32lg_stk3600

; change microcontroller
board_build.mcu = efm32lg990f256

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

EFM32LG-STK3600 Leopard Gecko supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:
PlatformIO Documentation, Release 5.0.5a1

```ini
[env:efm32lg_stk3600]
platform = siliconlabsefm32
board = efm32lg_stk3600
upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

EFM32LG-STK3600 Leopard Gecko has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

**EFM32WG-STK3800 Wonder Gecko**

**Contents**

- **EFM32WG-STK3800 Wonder Gecko**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform Silicon Labs EFM32: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFM32WG990F256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>

Configuration

Please use efm32wg_stk3800 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:efm32wg_stk3800]
platform = siliconlabsefm32
board = efm32wg_stk3800
```

You can override default EFM32WG-STK3800 Wonder Gecko settings per build environment using board_*** option, where *** is a JSON object path from board manifest efm32wg_stk3800.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:efm32wg_stk3800]
platform = siliconlabsefm32
board = efm32wg_stk3800

; change microcontroller
board_build.mcu = efm32wg990f256

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

EFM32WG-STK3800 Wonder Gecko supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:efm32wg_stk3800]
platform = siliconlabsefm32
board = efm32wg_stk3800

upload_protocol = mbed
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

EFM32WG-STK3800 Wonder Gecko has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

**EFM32ZG-STK3200 Zero Gecko**

**Contents**

- **EFM32ZG-STK3200 Zero Gecko**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Silicon Labs EFM32*: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.
### Configuration

Please use `efm32zg_stk3200` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:efm32zg_stk3200]
platform = siliconlabsefm32
board = efm32zg_stk3200
```

You can override default EFM32ZG-STK3200 Zero Gecko settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `efm32zg_stk3200.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:efm32zg_stk3200]
platform = siliconlabsefm32
board = efm32zg_stk3200

; change microcontroller
board_build.mcu = efm32zg222f32

; change MCU frequency
board_build.f_cpu = 24000000L
```

### Uploading

EFM32ZG-STK3200 Zero Gecko supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:efm32zg_stk3200]
platform = siliconlabsefm32
board = efm32zg_stk3200

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

EFM32ZG-STK3200 Zero Gecko has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

SLSTK3400A USB-enabled Happy Gecko

Contents

- SLSTK3400A USB-enabled Happy Gecko
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Silicon Labs EFM32: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFM32HG322F64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>25MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>
Configuration

Please use `efm32hg_stk3400` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:efm32hg_stk3400]
platform = siliconlabsefm32
board = efm32hg_stk3400
```

You can override default SLSTK3400A USB-enabled Happy Gecko settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `efm32hg_stk3400.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:efm32hg_stk3400]
platform = siliconlabsefm32
board = efm32hg_stk3400

; change microcontroller
board_build.mcu = efm32hg322f64

; change MCU frequency
board_build.f_cpu = 25000000L
```

Uploading

SLSTK3400A USB-enabled Happy Gecko supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:efm32hg_stk3400]
platform = siliconlabsefm32
board = efm32hg_stk3400

upload_protocol = mbed
```

Debugging

*Debugging* - "1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

SLSTK3400A USB-enabled Happy Gecko has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
## Compatible Tools

<table>
<thead>
<tr>
<th></th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
</tbody>
</table>

## SLSTK3401A Pearl Gecko PG1

### Contents

- SLSTK3401A Pearl Gecko PG1
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **Silicon Labs EFM32**: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFM32PG1B200F256GM48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>40MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>

### Configuration

Please use `efm32pg_stk3401` ID for `board` option in “`platformio.ini` (Project Configuration File):"
You can override default SLSTK3401A Pearl Gecko PG1 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `efm32pg_stk3401.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:efm32pg_stk3401]
platform = siliconlabsefm32
board = efm32pg_stk3401

; change microcontroller
board_build.mcu = efm32pg1b200f256gm48

; change MCU frequency
board_build.f_cpu = 40000000L
```

### Uploading

SLSTK3401A Pearl Gecko PG1 supports the next uploading protocols:

- `blackmagic`
- `jlink`
- `mbed`

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:efm32pg_stk3401]
platform = siliconlabsefm32
board = efm32pg_stk3401

upload_protocol = mbed
```

### Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

SLSTK3401A Pearl Gecko PG1 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

SLSTK3701A Giant Gecko S1

Contents

- **SLSTK3701A Giant Gecko S1**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *Silicon Labs EFM32*: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFM32GG11B820F2048GL192</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>

Configuration

Please use `efm32gg11_stk3701` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```
[env:efm32gg11_stk3701]
platform = siliconlabsefm32
board = efm32gg11_stk3701
```

You can override default SLSTK3701A Giant Gecko S1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `efm32gg11_stk3701.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

SLSTK3701A Giant Gecko S1 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:efm32gg11_stk3701]
platform = siliconlabsefm32
board = efm32gg11_stk3701
upload_protocol = mbed
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

SLSTK3701A Giant Gecko S1 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT

Contents

- Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Silicon Labs EFM32: Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFR32MG12P432F1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>40MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>

Configuration

Please use `tb_sense_12` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:tb_sense_12]
platform = siliconlabsefm32
board = tb_sense_12
```

You can override default Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `tb_sense_12.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.
Uploading

Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```
[env:tb_sense_12]
platform = siliconlabsefm32
board = tb_sense_12

upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” *(Project Configuration File)*.

Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

1.12.28 ST STM32

1Bitsy

Contents

- 1Bitsy
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F415RGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>1BitSquared</td>
</tr>
</tbody>
</table>

Configuration

Please use `1bitsy_stm32f415rgt` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:1bitsy_stm32f415rgt]
platform = ststm32
board = 1bitsy_stm32f415rgt
```

You can override default 1Bitsy settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `1bitsy_stm32f415rgt.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
[env:1bitsy_stm32f415rgt]
platform = ststm32
board = 1bitsy_stm32f415rgt

; change microcontroller
board_build.mcu = stm32f415rgt

; change MCU frequency
board_build.f_cpu = 168000000L

Uploading

1Bitsy supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is blackmagic

You can change upload protocol using upload_protocol option:

[env:1bitsy_stm32f415rgt]
platform = ststm32
board = 1bitsy_stm32f415rgt

upload_protocol = blackmagic

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in *platformio.ini* (Project Configuration File).

1Bitsy does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

32F412GDISCOVERY

Contents

- 32F412GDISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use disco_f412zg ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_f412zg]
platform = ststm32
board = disco_f412zg
```

You can override default 32F412GDISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_f412zg.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:disco_f412zg]
platform = ststm32
board = disco_f412zg

; change microcontroller
board_build.mcu = stm32f412zgt6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

32F412GDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:disco_f412zg]
platform = ststm32
board = disco_f412zg

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
32F412GDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

**32F723EDISCOVERY**

**Contents**

- 32F723EDISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F723IEK6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_f723ie ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_f723ie]
platform = ststm32
board = disco_f723ie
```

You can override default 32F723EDISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_f723ie.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_f723ie]
platform = ststm32
board = disco_f723ie

; change microcontroller
board_build.mcu = stm32f723iek6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

32F723EDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_f723ie]
platform = ststm32
board = disco_f723ie

upload_protocol = stlink
```
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

32F723EDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

**3D Printer Controller**

**Contents**

- 3D Printer Controller
  - Hardware
  - Configuration
  - Uploading
  - Debugging
### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Armed</td>
</tr>
</tbody>
</table>

### Configuration

Please use `armed_v1` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:armed_v1]
platform = ststm32
board = armed_v1
```

You can override default 3D Printer Controller settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `armed_v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:armed_v1]
platform = ststm32
board = armed_v1

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

3D Printer Controller supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:armed_v1]
platform = ststm32
board = armed_v1
upload_protocol = stlink

Debugging

*Warning:* You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

3D Printer Controller does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

3D Printer control board
Contents

- 3D Printer control board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F446RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RUMBA</td>
</tr>
</tbody>
</table>

Configuration

Please use `rumba32_f446ve` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:rumba32_f446ve]
platform = ststm32
board = rumba32_f446ve
```

You can override default 3D Printer control board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `rumba32_f446ve.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:rumba32_f446ve]
platform = ststm32
board = rumba32_f446ve

; change microcontroller
board_build.mcu = stm32f446ret6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

3D Printer control board supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• serial
• stlink
Default protocol is stlink
You can change upload protocol using upload_protocol option:

```
[env:rumba32_f446ve]
platform = ststm32
board = rumba32_f446ve
upload_protocol = stlink
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using debug_tool option in "platformio.ini" (Project Configuration File).

3D Printer control board does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

3D printer controller

Contents

- 3D printer controller
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F765VIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RemRam</td>
</tr>
</tbody>
</table>

Configuration

Please use `remram_v1` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default 3D printer controller settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `remram_v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:remram_v1]
platform = ststm32
board = remram_v1

; change microcontroller
board_build.mcu = stm32f765vit6

; change MCU frequency
board_build.f_cpu = 216000000L
```

### Uploading

3D printer controller supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:remram_v1]
platform = ststm32
board = remram_v1

upload_protocol = stlink
```

### Debugging

*Debugging*—“1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.

3D printer controller has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
*Black Magic Probe* |  |  
*CMSIS-DAP* |  |  
*J-LINK* |  |  
*ST-LINK* | Yes | Yes

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

## 3DP001V1 Evaluation board for 3D printer

### Contents

- 3DP001V1 Evaluation board for 3D printer
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
**Microcontroller** | STM32F401VGT6  
---|---
**Frequency** | 84MHz  
**Flash** | 512KB  
**RAM** | 96KB  
**Vendor** | ST

**Configuration**

Please use `st3dp001_eval` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:st3dp001_eval]
platform = ststm32
board = st3dp001_eval
```

You can override default 3DP001V1 Evaluation board for 3D printer settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `st3dp001_eval.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:st3dp001_eval]
platform = ststm32
board = st3dp001_eval

; change microcontroller
board_build.mcu = stm32f401vgt6

; change MCU frequency
board_build.f_cpu = 84000000L
```

**Uploading**

3DP001V1 Evaluation board for 3D printer supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:st3dp001_eval]
platform = ststm32
board = st3dp001_eval

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

3DP001V1 Evaluation board for 3D printer has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STMLIB</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

96Boards Argonkey (STEVAL-MKI187V1)

Contents

- 96Boards Argonkey (STEVAL-MKI187V1)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
PlatformIO Documentation, Release 5.0.5a1

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>96Boards</td>
</tr>
</tbody>
</table>

Configuration

Please use `b96b_argonkey` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:b96b_argonkey]
platform = ststm32
board = b96b_argonkey
```

You can override default 96Boards Argonkey (STEV-AL-MKI187V1) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `b96b_argonkey.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:b96b_argonkey]
platform = ststm32
board = b96b_argonkey

; change microcontroller
board_build.mcu = stm32f412cg

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

96Boards Argonkey (STEV-AL-MKI187V1) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

96Boards Argonkey (STEV AL-MKI187V1) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrtOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

96Boards B96B-F446VE
• 96Boards B96B-F446VE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F446VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>96Boards</td>
</tr>
</tbody>
</table>

Configuration

Please use `b96b_f446ve` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:b96b_f446ve]
platform = ststm32
board = b96b_f446ve
```

You can override default 96Boards B96B-F446VE settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `b96b_f446ve.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:b96b_f446ve]
platform = ststm32
board = b96b_f446ve

; change microcontroller
board_build.mcu = stm32f446vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

96Boards B96B-F446VE supports the next uploading protocols:

- blackmagic
- cmsis-dap
• jlink
• mbed
• stlink

Default protocol is **stlink**

You can change upload protocol using *upload_protocol* option:

```ini
[env:b96b_f446ve]
platform = ststm32
board = b96b_f446ve
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in *“platformio.ini” (Project Configuration File)*.

96Boards B96B-F446VE has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) and M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

96Boards Neonkey

- 96Boards Neonkey
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
<td>96Boards</td>
</tr>
</tbody>
</table>
Configuration

Please use b96b_aerocore2 ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:b96b_aerocore2]
platform = stm32
board = b96b_aerocore2
```

You can override default 96Boards Neonkey settings per build environment using board_*** option, where *** is a JSON object path from board manifest b96b_aerocore2.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:b96b_aerocore2]
platform = stm32
board = b96b_aerocore2

; change microcontroller
board_build.mcu = stm32f427vit6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

96Boards Neonkey supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is dfu

You can change upload protocol using upload_protocol option:

```ini
[env:b96b_aerocore2]
platform = stm32
board = b96b_aerocore2

upload_protocol = dfu
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
96Boards Neonkey does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM-SIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOs</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

**96Boards Neonkey**

**Contents**

- 96Boards Neonkey
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>96Boards</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `b96b_neonkey` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:b96b_neonkey]
platform = ststm32
board = b96b_neonkey
```

You can override default 96Boards Neonkey settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `b96b_neonkey.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:b96b_neonkey]
platform = ststm32
board = b96b_neonkey

; change microcontroller
board_build.mcu = stm32f411ce

; change MCU frequency
board_build.f_cpu = 100000000L
```

**Uploading**

96Boards Neonkey supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:b96b_neonkey]
platform = ststm32
board = b96b_neonkey

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

96Boards Neonkey does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

Adafruit Feather STM32F405

- Adafruit Feather STM32F405
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F405RGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `adafruit_feather_f405` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:adafruit_feather_f405]
platform = ststm32
board = adafruit_feather_f405
```

You can override default Adafruit Feather STM32F405 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `adafruit_feather_f405.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:adafruit_feather_f405]
platform = ststm32
board = adafruit_feather_f405

; change microcontroller
board_build.mcu = stm32f405rgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

Adafruit Feather STM32F405 supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:adafruit_feather_f405]
platform = ststm32
board = adafruit_feather_f405
upload_protocol = stlink

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Adafruit Feather STM32F405 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

AfroFlight Rev5 (8MHz)
Contents

- AfroFlight Rev5 (8MHz)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>AfroFlight</td>
</tr>
</tbody>
</table>

Configuration

Please use afroflight_f103cb ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:afroflight_f103cb]
platform = ststm32
board = afroflight_f103cb
```

You can override default AfroFlight Rev5 (8MHz) settings per build environment using board_*** option, where *** is a JSON object path from board manifest afroflight_f103cb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:afroflight_f103cb]
platform = ststm32
board = afroflight_f103cb

; change microcontroller
board_build.mcu = stm32f103cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

AfroFlight Rev5 (8MHz) supports the next uploading protocols:
• blackmagic
• cmsis-dap
• dfu
• jlink
• serial
• stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:afroflight_f103cb]
platform = ststm32
board = afroflight_f103cb
upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

AfroFlight Rev5 (8MHz) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Armstrap Eagle 1024

Contents

- Armstrap Eagle 1024
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F417VGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Armstrap</td>
</tr>
</tbody>
</table>

Configuration

Please use `armstrap_eagle1024` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default Armstrap Eagle 1024 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `armstrap_eagle1024.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:armstrap_eagle1024]
platform = ststm32
board = armstrap_eagle1024

; change microcontroller
board_build.mcu = stm32f417vgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

# Uploading

Armstrap Eagle 1024 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is **blackmagic**

You can change upload protocol using `upload_protocol` option:

```ini
[env:armstrap_eagle1024]
platform = ststm32
board = armstrap_eagle1024

upload_protocol = blackmagic
```

# Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Armstrap Eagle 1024 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
## Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

## Armstrap Eagle 2048

### Contents

- Armstrap Eagle 2048
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
Microcontroller | STM32F427VIT6  
--- | ---  
Frequency | 168MHz  
Flash | 1.99MB  
RAM | 256KB  
Vendor | Armstrap

**Configuration**

Please use `armstrap_eagle2048` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:armstrap_eagle2048]
platform = ststm32
board = armstrap_eagle2048
```

You can override default Armstrap Eagle 2048 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `armstrap_eagle2048.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:armstrap_eagle2048]
platform = ststm32
board = armstrap_eagle2048

; change microcontroller
board_build.mcu = stm32f427vit6

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

Armstrap Eagle 2048 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is blackmagic

You can change upload protocol using `upload_protocol` option:

```
[env:armstrap_eagle2048]
platform = ststm32
board = armstrap_eagle2048

upload_protocol = blackmagic
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Armstrap Eagle 2048 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Armstrap Eagle 512

Contents

- Armstrap Eagle 512
  - Hardware
  - Configuration
  - Uploading
  - Debugging
Hardware

Platform *STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Armstrap</td>
</tr>
</tbody>
</table>

Configuration

Please use armstrap_eagle512 ID for *board* option in “platformio.ini” (Project Configuration File):

```
[env:armstrap_eagle512]
platform = ststm32
board = armstrap_eagle512
```

You can override default Armstrap Eagle 512 settings per build environment using *board_*** option, where *** is a JSON object path from board manifest armstrap_eagle512.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:armstrap_eagle512]
platform = ststm32
board = armstrap_eagle512

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

Armstrap Eagle 512 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is blackmagic

You can change upload protocol using *upload_protocol* option:
[env:armstrap_eagle512]
platform = ststm32
board = armstrap_eagle512
upload_protocol = blackmagic

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Armstrap Eagle 512 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | Yes
CMSIS-DAP
J-LINK
ST-LINK

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
Black STM32F407VE

Contents

• Black STM32F407VE
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use black_f407ve ID for board option in “platformio.ini” (Project Configuration File):

```
[env:black_f407ve]
platform = ststm32
board = black_f407ve
```

You can override default Black STM32F407VE settings per build environment using board_*** option, where *** is a JSON object path from board manifest black_f407ve.json. For example, board_build.mcu, board_build. f_cpu, etc.

```
[env:black_f407ve]
platform = ststm32
board = black_f407ve

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```
Uploading

Black STM32F407VE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:black_f407ve]
platform = ststm32
board = black_f407ve

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Black STM32F407VE does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Black STM32F407VG

Contents

- Black STM32F407VG
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use `black_f407vg` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:black_f407vg]
platform = ststm32
board = black_f407vg
```

You can override default Black STM32F407VG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `black_f407vg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:black_f407vg]
platform = ststm32
board = black_f407vg

; change microcontroller
board_build.mcu = stm32f407vgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

Black STM32F407VG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:black_f407vg]
platform = ststm32
board = black_f407vg

upload_protocol = stlink
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).
Black STM32F407VG does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**Black STM32F407ZE**

**Contents**

- **Black STM32F407ZE**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `black_f407ze` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:black_f407ze]
platform = ststm32
board = black_f407ze
```

You can override default Black STM32F407ZE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `black_f407ze.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:black_f407ze]
platform = ststm32
board = black_f407ze

; change microcontroller
board_build.mcu = stm32f407zet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

Black STM32F407ZE supports the next uploading protocols:
- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:black_f407ze]
platform = ststm32
board = black_f407ze

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

---

**1.12. Boards** 1875
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Black STM32F407ZE does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

---

### Black STM32F407ZG

#### Contents

- **Black STM32F407ZG**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
# Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

## Configuration

Please use `black_f407zg` ID for `board` option in `platformio.ini` (Project Configuration File):

```
[env:black_f407zg]
platform = ststm32
board = black_f407zg
```

You can override default Black STM32F407ZG settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `black_f407zg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:black_f407zg]
platform = ststm32
board = black_f407zg

; change microcontroller
board_build.mcu = stm32f407zgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

## Uploading

Black STM32F407ZG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
```
[env: black_f407zg]
platform = ststm32
board = black_f407zg
upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Black STM32F407ZG does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of the external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

BlackPill F103C8
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `blackpill_f103c8` ID for `board` option in **“platformio.ini” (Project Configuration File)**:

```ini
[env:blackpill_f103c8]
platform = ststm32
board = blackpill_f103c8
```

You can override default BlackPill F103C8 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `blackpill_f103c8.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```ini
[env:blackpill_f103c8]
platform = ststm32
board = blackpill_f103c8

; change microcontroller
board_build.mcu = stm32f103c8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

BlackPill F103C8 supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• serial
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:blackpill_f103c8]
platform = ststm32
board = blackpill_f103c8
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

BlackPill F103C8 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

BlackPill F103C8 (128k)

Contents

• BlackPill F103C8 (128k)
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>
Configuration

Please use `blackpill_f103c8_128` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:blackpill_f103c8_128]
platform = ststm32
board = blackpill_f103c8_128
```

You can override default BlackPill F103C8 (128k) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `blackpill_f103c8_128.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:blackpill_f103c8_128]
platform = ststm32
board = blackpill_f103c8_128

; change microcontroller
board_build.mcu = stm32f103c8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

BlackPill F103C8 (128k) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:blackpill_f103c8_128]
platform = ststm32
board = blackpill_f103c8_128

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
BlackPill F103C8 (128k) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### BlackPill F303CC

#### Contents

- **BlackPill F303CC**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F303CCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>40KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RobotDyn</td>
</tr>
</tbody>
</table>

Configuration

Please use `robotdyn_blackpill_f303cc` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:robotdyn_blackpill_f303cc]
platform = ststm32
board = robotdyn_blackpill_f303cc
```

You can override default BlackPill F303CC settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `robotdyn_blackpill_f303cc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:robotdyn_blackpill_f303cc]
platform = ststm32
board = robotdyn_blackpill_f303cc

; change microcontroller
board_build.mcu = stm32f303cct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

BlackPill F303CC supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:robotdyn_blackpill_f303cc]
platform = ststm32
board = robotdyn_blackpill_f303cc

upload_protocol = stlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

BlackPill F303CC does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

BlackPill F401CC

Contents

- BlackPill F401CC
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CCU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `blackpill_f401cc` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:blackpill_f401cc]
platform = ststm32
board = blackpill_f401cc
```

You can override default BlackPill F401CC settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `blackpill_f401cc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:blackpill_f401cc]
platform = ststm32
board = blackpill_f401cc

; change microcontroller
board_build.mcu = stm32f401ccu6

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

BlackPill F401CC supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `dfu`
- `jlink`
- `serial`
- `stlink`
Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```
[env:blackpill_f401cc]
platform = ststm32
board = blackpill_f401cc
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

BlackPill F401CC does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
BlackPill F401CE

Contents

- BlackPill F401CE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CEU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use blackpill_f401ce ID for board option in “platformio.ini” (Project Configuration File):

```
[env:blackpill_f401ce]
platform = ststm32
board = blackpill_f401ce
```

You can override default BlackPill F401CE settings per build environment using board_*** option, where *** is a JSON object path from board manifest blackpill_f401ce.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:blackpill_f401ce]
platform = ststm32
board = blackpill_f401ce

; change microcontroller
board_build.mcu = stm32f401ceu6

; change MCU frequency
board_build.f_cpu = 84000000L
```
Uploading

BlackPill F401CE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:blackpill_f401ce]
platform = ststm32
board = blackpill_f401ce
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

BlackPill F401CE does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, TI Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

Blue STM32F407VE Mini

**Contents**

- Blue STM32F407VE Mini
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `blue_f407ve_mini` ID for **board** option in **platformio.ini** (Project Configuration File):
You can override default Blue STM32F407VE Mini settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `blue_f407ve_mini.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:blue_f407ve_mini]
platform = ststm32
board = blue_f407ve_mini

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

Blue STM32F407VE Mini supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:blue_f407ve_mini]
platform = ststm32
board = blue_f407ve_mini

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

Blue STM32F407VE Mini does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### Blue Pill F103C6

#### Contents

- Blue Pill F103C6
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C6T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `bluepill_f103c6` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:bluepill_f103c6]
platform = ststm32
board = bluepill_f103c6
```

You can override default BluePill F103C6 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `bluepill_f103c6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:bluepill_f103c6]
platform = ststm32
board = bluepill_f103c6

; change microcontroller
board_build.mcu = stm32f103c6t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

BluePill F103C6 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:bluepill_f103c6]
platform = ststm32
board = bluepill_f103c6

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” *(Project Configuration File)*.

BluePill F103C6 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**BluePill F103C8**

**Contents**

- **BluePill F103C8**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `bluepill_f103c8` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:bluepill_f103c8]
platform = ststm32
board = bluepill_f103c8
```

You can override default BluePill F103C8 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bluepill_f103c8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:bluepill_f103c8]
platform = ststm32
board = bluepill_f103c8

; change microcontroller
board_build.mcu = stm32f103c8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

BluePill F103C8 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:bluepill_f103c8]
platform = stm32
board = bluepill_f103c8
upload_protocol = stlink

---

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

BluePill F103C8 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
BluePill F103C8 (128k)

Contents

- BluePill F103C8 (128k)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use bluepill_f103c8_128k ID for board option in “platformio.ini” (Project Configuration File):

```
[env:bluepill_f103c8_128k]
platform = ststm32
board = bluepill_f103c8_128k
```

You can override default BluePill F103C8 (128k) settings per build environment using board_*** option, where *** is a JSON object path from board manifest bluepill_f103c8_128k.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:bluepill_f103c8_128k]
platform = ststm32
board = bluepill_f103c8_128k

; change microcontroller
board_build.mcu = stm32f103c8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

BluePill F103C8 (128k) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:bluepill_f103c8_128k]
platform = ststm32
board = bluepill_f103c8_128k

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

BluePill F103C8 (128k) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Cicada-L082CZ

Contents

- Cicada-L082CZ
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L082CZY6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Tiera Corporation</td>
</tr>
</tbody>
</table>
Configuration

Please use `cicada_l082cz` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:cicada_l082cz]
platform = stm32
board = cicada_l082cz
```

You can override default Cicada-L082CZ settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cicada_l082cz.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:cicada_l082cz]
platform = stm32
board = cicada_l082cz

; change microcontroller
board_build.mcu = stm32l082czy6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Cicada-L082CZ supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- stlink

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:

```ini
[env:cicada_l082cz]
platform = stm32
board = cicada_l082cz

upload_protocol = dfu
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
Cicada-L082CZ does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

Core board F401RCT6

Contents

- Core board F401RCT6
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401RCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use coreboard_f401rc ID for board option in “platformio.ini” (Project Configuration File):

```
[env:coreboard_f401rc]
platform = ststm32
board = coreboard_f401rc
```

You can override default Core board F401RCT6 settings per build environment using board_*** option, where *** is a JSON object path from board manifest coreboard_f401rc.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:coreboard_f401rc]
platform = ststm32
board = coreboard_f401rc

; change microcontroller
board_build.mcu = stm32f401rct6

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

Core board F401RCT6 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- hid
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:coreboard_f401rc]
platform = ststm32
board = coreboard_f401rc

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Core board F401RCT6 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**Cricket-L082CZ**

**Contents**

- Cricket-L082CZ
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L082CZY6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Tlera Corporation</td>
</tr>
</tbody>
</table>

### Configuration

Please use cricket_1082cz ID for board option in “platformio.ini” (Project Configuration File):

```
[env:cricket_1082cz]
platform = ststm32
board = cricket_1082cz
```

You can override default Cricket-L082CZ settings per build environment using board_*** option, where *** is a JSON object path from board manifest cricket_1082cz.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:cricket_1082cz]
platform = ststm32
board = cricket_1082cz

; change microcontroller
board_build.mcu = stm32l082czy6

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Uploading

Cricket-L082CZ supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- stlink

Default protocol is dfu

You can change upload protocol using upload_protocol option:

```
[env:cricket_1082cz]
platform = ststm32
board = cricket_1082cz

upload_protocol = dfu
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in **“platformio.ini” (Project Configuration File)**.

Cricket-L082CZ does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

Demo F030F4

**Contents**

- Demo F030F4
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F030F4P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use demo_f030f4 ID for `board` option in “platformio.ini (Project Configuration File):

```
[env:demo_f030f4]
platform = ststm32
board = demo_f030f4
```

You can override default Demo F030F4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest demo_f030f4.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:demo_f030f4]
platform = ststm32
board = demo_f030f4

; change microcontroller
board_build.mcu = stm32f030f4p6

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

Demo F030F4 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:demo_f030f4]
platform = ststm32
board = demo_f030f4

upload_protocol = stlink
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Demo F030F4 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Econode-L082CZ

Contents

- Econode-L082CZ
  - Hardware
  - Configuration
  - Uploading
PlatformIO Documentation, Release 5.0.5a1

– Debugging
– Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L082CZY6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Econode</td>
</tr>
</tbody>
</table>

Configuration

Please use econode_1082cz ID for board option in “platformio.ini” (Project Configuration File):

```
[env:econode_l082cz]
platform = ststm32
board = econode_l082cz
```

You can override default Econode-L082CZ settings per build environment using board_*** option, where *** is a JSON object path from board manifest econode_l082cz.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:econode_l082cz]
platform = ststm32
board = econode_l082cz

; change microcontroller
board_build.mcu = stm32l082czy6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Econode-L082CZ supports the next uploading protocols:
- blackmagic
- dfu
- jlink
- stlink

Default protocol is dfu

You can change upload protocol using upload_protocol option:
[env:econode_l082cz]
platform = stm32
board = econode_l082cz
upload_protocol = dfu

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Econode-L082CZ does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

Electrosmith Daisy

Contents

- Electrosmith Daisy
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32H750IBK6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Electrosmith</td>
</tr>
</tbody>
</table>

Configuration

Please use `electrosmith_daisy` ID for *board* option in “platformio.ini” (Project Configuration File):

```
[env:electrosmith_daisy]
platform = ststm32
board = electrosmith_daisy
```

You can override default Electrosmith Daisy settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `electrosmith_daisy.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:electrosmith_daisy]
platform = ststm32
board = electrosmith_daisy

; change microcontroller
board_build.mcu = stm32h750ibk6

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

Electrosmith Daisy supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `jlink`
- `mbed`
- `stlink`

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:electrosmith_daisy]
platform = stm32
board = electrosmith_daisy
upload_protocol = stlink

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Electrosmith Daisy does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

Espotel LoRa Module

- Espotel LoRa Module
  - Hardware
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Espotel</td>
</tr>
</tbody>
</table>

Configuration

Please use `elmo_f411re` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:elmo_f411re]
platform = ststm32
board = elmo_f411re
```

You can override default Espotel LoRa Module settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `elmo_f411re.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:elmo_f411re]
platform = ststm32
board = elmo_f411re

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

Espotel LoRa Module supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink
Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:elmo_f411re]
platform = ststm32
board = elmo_f411re
upload_protocol = stlink
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Espotel LoRa Module does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-Link</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
F407VG

Contents

• F407VG
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Diymore</td>
</tr>
</tbody>
</table>

Configuration

Please use diymore_f407vgt ID for board option in “platformio.ini” (Project Configuration File):

```
[env:diymore_f407vgt]
platform = ststm32
board = diymore_f407vgt
```

You can override default F407VG settings per build environment using board_*** option, where *** is a JSON object path from board manifest diymore_f407vgt.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:diymore_f407vgt]
platform = ststm32
board = diymore_f407vgt

; change microcontroller
board_build.mcu = stm32f407vgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```
Uploading

F407VG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:diymore_f407vgt]
platform = ststm32
board = diymore_f407vgt
upload_protocol = stlink
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

F407VG does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

FK407M1

Contents

- FK407M1
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use fk407m1 ID for board option in “platformio.ini” (Project Configuration File):
You can override default FK407M1 settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `fk407m1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:fk407m1]
platform = ststm32
board = fk407m1

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

FK407M1 supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `jlink`
- `serial`
- `stlink`

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:fk407m1]
platform = ststm32
board = fk407m1

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File).*

FK407M1 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

**Name** | **Description**
---|---
Arduino | Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS | The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
STM32Cube | STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.
libOpenCM3 | The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.

### FYSETC S6

**Contents**

- **FYSETC S6**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.


<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F446VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>FYSETC</td>
</tr>
</tbody>
</table>

### Configuration

Please use `fysetc_s6` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:fysetc_s6]
platform = ststm32
board = fysetc_s6
```

You can override default FYSETC S6 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `fysetc_s6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:fysetc_s6]
platform = ststm32
board = fysetc_s6

; change microcontroller
board_build.mcu = stm32f446vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

FYSETC S6 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:fysetc_s6]
platform = ststm32
board = fysetc_s6

upload_protocol = stlink
```
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

FYSETC S6 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Tiva, Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Gnat-L082CZ

Contents

- Gnat-L082CZ
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L082CZY6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Tiera Corporation</td>
</tr>
</tbody>
</table>

Configuration

Please use `gnat_l082cz` ID for *board* option in “`platformio.ini`” (*Project Configuration File*):

```ini
[env:gnat_l082cz]
platform = ststm32
board = gnat_l082cz
```

You can override default Gnat-L082CZ settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `gnat_l082cz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:gnat_l082cz]
platform = ststm32
board = gnat_l082cz

; change microcontroller
board_build.mcu = stm32l082czy6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Gnat-L082CZ supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- stlink

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:
```
[env:gnat_1082cz]
platform = stm32
board = gnat_1082cz
upload_protocol = dfu
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

Gnat-L082CZ does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

### Grasshopper-L082CZ

**Contents**

- Grasshopper-L082CZ
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L082CZY6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Tlera Corporation</td>
</tr>
</tbody>
</table>

Configuration

Please use `grasshopper_l082cz` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:grasshopper_l082cz]
platform = stm32
board = grasshopper_l082cz
```

You can override default Grasshopper-L082CZ settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `grasshopper_l082cz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:grasshopper_l082cz]
platform = stm32
board = grasshopper_l082cz

; change microcontroller
board_build.mcu = stm32l082czy6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Grasshopper-L082CZ supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- stlink

Default protocol is dfu

You can change upload protocol using `upload_protocol` option:

```
[env:grasshopper_l082cz]
platform = stm32
board = grasshopper_l082cz
```

(continues on next page)
debugging

- “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Grasshopper-L082CZ does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

L476DMW1K

Contents

- L476DMW1K
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L476VGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>rhomb.io</td>
</tr>
</tbody>
</table>

Configuration

Please use rhombio_l476dmw1k ID for board option in "platformio.ini" (Project Configuration File):

```
[env:rhombio_l476dmw1k]
platform = ststm32
board = rhombio_l476dmw1k
```

You can override default L476DMW1K settings per build environment using board_*** option, where *** is a JSON object path from board manifest rhombio_l476dmw1k.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:rhombio_l476dmw1k]
platform = ststm32
board = rhombio_l476dmw1k

; change microcontroller
board_build.mcu = stm32l476vgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

L476DMW1K supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is cmsis-dap

You can change upload protocol using upload_protocol option:

```
[env:rhombio_l476dmw1k]
platform = ststm32
board = rhombio_l476dmw1k
```

(continues on next page)
upload_protocol = cmsis-dap

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

L476DMW1K has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>J-LINK</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ST-LINK</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CMSIS</strong></td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td><strong>libopencm3</strong></td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

M200 V2

**Contents**

- **M200 V2**
Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F070CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Malyan</td>
</tr>
</tbody>
</table>

**Configuration**

Please use malyanm200_f070cb ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:malyanm200_f070cb]
platform = ststm32
board = malyanm200_f070cb
```

You can override default M200 V2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest malyanm200_f070cb.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:malyanm200_f070cb]
platform = ststm32
board = malyanm200_f070cb

; change microcontroller
board_build.mcu = stm32f070cbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

M200 V2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
• serial
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:malyann200_f070cb]
platform = ststm32
board = malyann200_f070cb
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

M200 V2 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Tī Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
M300

Contents

- M300
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F070CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Malyan</td>
</tr>
</tbody>
</table>

Configuration

Please use malyanm300_f070cb ID for board option in “platformio.ini” (Project Configuration File):

```
[env:malyanm300_f070cb]
platform = ststm32
board = malyanm300_f070cb
```

You can override default M300 settings per build environment using board_*** option, where *** is a JSON object path from board manifest malyanm300_f070cb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:malyanm300_f070cb]
platform = ststm32
board = malyanm300_f070cb

; change microcontroller
board_build.mcu = stm32f070cbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```
Uploading

M300 supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:malyanm300_f070cb]
platform = ststm32
board = malyanm300_f070cb
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

M300 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

MKR Sharky

Contents

- **MKR Sharky**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32WB55CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192,000KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Midatronics</td>
</tr>
</tbody>
</table>

Configuration

Please use `mkr_sharky` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):
You can override default MKR Sharky settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mkr_sharky.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mkr_sharky]
platform = ststm32
board = mkr_sharky

; change microcontroller
board_build.mcu = stm32wb55cg

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

MKR Sharky supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- mbed
- serial

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:mkr_sharky]
platform = ststm32
board = mkr_sharky

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

MKR Sharky does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

MTS Dragonfly

Contents

- MTS Dragonfly
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MultiTech</td>
</tr>
</tbody>
</table>

Configuration

Please use mts_dragonfly_f411re ID for board option in "platformio.ini" (Project Configuration File):

```
[env:mts_dragonfly_f411re]
platform = ststm32
board = mts_dragonfly_f411re
```
You can override default MTS Dragonfly settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `mts_dragonfly_f411re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mts_dragonfly_f411re]
platform = ststm32
board = mts_dragonfly_f411re

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

MTS Dragonfly supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is mbed.

You can change upload protocol using `upload_protocol` option:

```ini
[env:mts_dragonfly_f411re]
platform = ststm32
board = mts_dragonfly_f411re

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

MTS Dragonfly does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | |
CMSIS-DAP | | |
J-LINK | | |
ST-LINK | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td><strong>libopencm3</strong></td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Malyan M200 V1

Contents

- **Malyan M200 V1**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Malyan</td>
</tr>
</tbody>
</table>
Configuration

Please use `malyanm200_f103cb` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:malyanm200_f103cb]
platform = ststm32
board = malyanm200_f103cb
```

You can override default Malyan M200 V1 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `malyanm200_f103cb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:malyanm200_f103cb]
platform = ststm32
board = malyanm200_f103cb

; change microcontroller
board_build.mcu = stm32f103cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Malyan M200 V1 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:malyanm200_f103cb]
platform = ststm32
board = malyanm200_f103cb

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
Malyan M200 V1 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**Maple**

**Contents**

- Maple
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
Microcontroller | STM32F103RBT6  
--- | ---  
Frequency | 72MHz  
Flash | 108KB  
RAM | 17KB  
Vendor | LeafLabs  

## Configuration

Please use maple ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:maple]
platform = ststm32
board = maple
```

You can override default Maple settings per build environment using `board_***` option, where *** is a JSON object path from board manifest maple.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:maple]
platform = ststm32
board = maple

; change microcontroller
board_build.mcu = stm32f103rbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

## Uploading

Maple supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- stlink

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:

```ini
[env:maple]
platform = ststm32
board = maple

upload_protocol = dfu
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Maple does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**Maple (RET6)**

**Contents**

- **Maple (RET6)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>48KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LeafLabs</td>
</tr>
</tbody>
</table>

Configuration

Please use `maple_ret6` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:maple_ret6]
platform = ststm32
board = maple_ret6
```

You can override default Maple (RET6) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `maple_ret6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:maple_ret6]
platform = ststm32
board = maple_ret6

; change microcontroller
board_build.mcu = stm32f103ret6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Maple (RET6) supports the next uploading protocols:

- `blackmagic`
- `cmsis-dap`
- `dfu`
- `jlink`
- `stlink`

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:
PlatformIO Documentation, Release 5.0.5a1

```
[env:maple_ret6]
platform = stm32
board = maple_ret6
upload_protocol = dfu
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Maple (RET6) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**Maple Mini Bootloader 2.0**
PlatformIO Documentation, Release 5.0.5a1

Contents

- *Maple Mini Bootloader 2.0*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>120KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LeafLabs</td>
</tr>
</tbody>
</table>

Configuration

Please use `maple_mini_b20` ID for `board` option in "`platformio.ini` (Project Configuration File):

```ini
[env:maple_mini_b20]
platform = ststm32
board = maple_mini_b20
```

You can override default Maple Mini Bootloader 2.0 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `maple_mini_b20.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:maple_mini_b20]
platform = ststm32
board = maple_mini_b20

; change microcontroller
board_build.mcu = stm32f103cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Maple Mini Bootloader 2.0 supports the next uploading protocols:
- blackmagic
- cmsis-dap
- dfu
- jlink
- stlink

Default protocol is dfu

You can change upload protocol using `upload_protocol` option:

```
[env:maple_mini_b20]
platform = ststm32
board = maple_mini_b20
upload_protocol = dfu
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Maple Mini Bootloader 2.0 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CM-SIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Maple Mini Original

Contents

- Maple Mini Original
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
<td>LeafLabs</td>
</tr>
</tbody>
</table>

Configuration

Please use `maple_mini_origin` ID for `board` option in “platformio.ini” *(Project Configuration File)*:
You can override default Maple Mini Original settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `maple_mini_origin.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:maple_mini_origin]
platform = ststm32
board = maple_mini_origin

; change microcontroller
board_build.mcu = stm32f103c8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

Maple Mini Original supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is dfu

You can change upload protocol using `upload_protocol` option:

```ini
[env:maple_mini_origin]
platform = ststm32
board = maple_mini_origin

upload_protocol = dfu
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.
Maple Mini Original does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**Mbed Connect Cloud**

**Contents**

- **Mbed Connect Cloud**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `mbed_connect_odin` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:mbed_connect_odin]
platform = ststm32
board = mbed_connect_odin
```

You can override default Mbed Connect Cloud settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mbed_connect_odin.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mbed_connect_odin]
platform = ststm32
board = mbed_connect_odin

; change microcontroller
board_build.mcu = ststm32f439ziy6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

Mbed Connect Cloud supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:mbed_connect_odin]
platform = ststm32
board = mbed_connect_odin

upload_protocol = stlink
```

### Debugging

*Debugging*- “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Mbed Connect Cloud has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tí Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Microduino Core STM32 to Flash

Contents

- Microduino Core STM32 to Flash
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>105.47KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16.60KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Microduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `microduino32_flash` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:microduino32_flash]
platform = ststm32
board = microduino32_flash
```

You can override default Microduino Core STM32 to Flash settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `microduino32_flash.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:microduino32_flash]
platform = ststm32
board = microduino32_flash

; change microcontroller
board_build.mcu = stm32f103cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

Microduino Core STM32 to Flash supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- stlink

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:

```ini
[env:microduino32_flash]
platform = ststm32
board = microduino32_flash

upload_protocol = dfu
```

1.12. Boards
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool option in “platformio.ini” (Project Configuration File)**.

Microduino Core STM32 to Flash does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Microsoft Azure IoT Development Kit (MXChip AZ3166)

- **Microsoft Azure IoT Development Kit (MXChip AZ3166)**
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MXChip</td>
</tr>
</tbody>
</table>

Configuration

Please use `mxchip_az3166` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:mxchip_az3166]
platform = ststm32
board = mxchip_az3166
don
```

You can override default Microsoft Azure IoT Development Kit (MXChip AZ3166) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `mxchip_az3166.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mxchip_az3166]
platform = ststm32
board = mxchip_az3166

don
; change microcontroller
board_build.mcu = stm32f412zgt6

don
; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

Microsoft Azure IoT Development Kit (MXChip AZ3166) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
[env:mxchip_az3166]
platform = stm32
board = mxchip_az3166
upload_protocol = stlink

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

Microsoft Azure IoT Development Kit (MXChip AZ3166) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

MultiTech mDot
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MultiTech</td>
</tr>
</tbody>
</table>

Configuration

Please use `mts_mdot_f405rg` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:mts_mdot_f405rg]
platform = ststm32
board = mts_mdot_f405rg
```

You can override default MultiTech mDot settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mts_mdot_f405rg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mts_mdot_f405rg]
platform = ststm32
board = mts_mdot_f405rg

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

MultiTech mDot supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• mbed
• stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:mts_mdot_f405rg]
platform = ststm32
board = mts_mdot_f405rg
upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *`platformio.ini` (Project Configuration File)*.

MultiTech mDot does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Tí Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
MultiTech mDot F411

Contents

- MultiTech mDot F411
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MultiTech</td>
</tr>
</tbody>
</table>

Configuration

Please use mts_mdot_f411re ID for board option in “platformio.ini” (Project Configuration File):

```
[env:mts_mdot_f411re]
platform = ststm32
board = mts_mdot_f411re
```

You can override default MultiTech mDot F411 settings per build environment using board_*** option, where *** is a JSON object path from board manifest mts_mdot_f411re.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:mts_mdot_f411re]
platform = ststm32
board = mts_mdot_f411re

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```
Uploading

MultiTech mDot F411 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:mts_mdot_f411re]
platform = ststm32
board = mts_mdot_f411re
upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

MultiTech mDot F411 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

MultiTech xDot

Contents

- *MultiTech xDot*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L151CCU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MultiTech</td>
</tr>
</tbody>
</table>

Configuration

Please use `xdot_l151cc` ID for *board* option in “`platformio.ini`” (Project Configuration File):

```
[env:xdot_l151cc]
platform = ststm32
board = xdot_l151cc
```
You can override default MultiTech xDot settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `xdot_l151cc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xdot_l151cc]
platform = ststm32
board = xdot_l151cc

; change microcontroller
board_build.mcu = stm32l151ccu6

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Uploading

MultiTech xDot supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:xdot_l151cc]
platform = ststm32
board = xdot_l151cc

upload_protocol = stlink
```

### Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

MultiTech xDot does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

N2+

Contents

- N2+
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F405RGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Netduino</td>
</tr>
</tbody>
</table>

Configuration

Please use netduino2plus ID for board option in “platformio.ini” (Project Configuration File):
You can override default N2+ settings per build environment using board_*** option, where *** is a JSON object path from board manifest netduino2plus.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:netduino2plus]
platform = ststm32
board = netduino2plus

; change microcontroller
board_build.mcu = stm32f405rgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

N2+ supports the next uploading protocols:

- dfu
- jlink
- stlink

Default protocol is dfu

You can change upload protocol using upload_protocol option:

```ini
[env:netduino2plus]
platform = ststm32
board = netduino2plus

upload_protocol = dfu
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

N2+ does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

NAMote72

Contents

- NAMote72
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L152RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Semtech</td>
</tr>
</tbody>
</table>

Configuration

Please use `mote_l152rc` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default NAMote72 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mote_l152rc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mote_l152rc]
platform = ststm32
board = mote_l152rc

; change microcontroller
board_build.mcu = stm32l152rc

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Uploading

NAMote72 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:mote_l152rc]
platform = ststm32
board = mote_l152rc

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

NAMote72 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
## Compatible Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Tivi Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

## Nucleo G071RB

### Contents

- **Nucleo G071RB**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `nucleo_g071rb` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_g071rb]
platform = ststm32
board = nucleo_g071rb
```

You can override default Nucleo G071RB settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_g071rb.json`. For example, `board_build.mcu, board_build. f_cpu, etc.:

```
[env:nucleo_g071rb]
platform = ststm32
board = nucleo_g071rb

; change microcontroller
board_build.mcu = stm32g071rbt6

; change MCU frequency
board_build.f_cpu = 64000000L
```

### Uploading

Nucleo G071RB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_g071rb]
platform = ststm32
board = nucleo_g071rb

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Nucleo G071RB does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Nucleo G431KB
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32G431KBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>170MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_g431kb ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:nucleo_g431kb]
platform = ststm32
board = nucleo_g431kb
```

You can override default Nucleo G431KB settings per build environment using *board_*** option, where *** is a JSON object path from board manifest nucleo_g431kb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_g431kb]
platform = ststm32
board = nucleo_g431kb

; change microcontroller
board_build.mcu = stm32g431kbt6

; change MCU frequency
board_build.f_cpu = 170000000L
```

Uploading

Nucleo G431KB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```ini
[env:nucleo_g431kb]
platform = ststm32
board = nucleo_g431kb
```

(continues on next page)
upload_protocol = mbed

### Debugging

*Debugging - “1-click” solution for debugging with a zero configuration.*

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

Nucleo G431KB does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>XTM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### Nucleo G431RB

#### Contents

- **Nucleo G431RB**
  - **Hardware**
PlatformIO Documentation, Release 5.0.5a1

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32G431RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>170MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `nucleo_g431rb` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_g431rb]
platform = ststm32
board = nucleo_g431rb
```

You can override default Nucleo G431RB settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_g431rb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_g431rb]
platform = ststm32
board = nucleo_g431rb

; change microcontroller
daemon = ststm32
board = nucleo_g431rb

; change MCU frequency
board_build.mcu = ststm32g431rbd6
board_build.f_cpu = 170000000L
```

Uploading

Nucleo G431RB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_g431rb]
platform = ststm32
board = nucleo_g431rb
upload_protocol = mbed
```

## Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Nucleo G431RB does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
Nucleo G474RE

Contents

- Nucleo G474RE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32G474RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>170MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_g474re ID for board option in “platformio.ini” (Project Configuration File):

```plaintext
[env:nucleo_g474re]
platform = ststm32
board = nucleo_g474re
```

You can override default Nucleo G474RE settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_g474re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:nucleo_g474re]
platform = ststm32
board = nucleo_g474re

; change microcontroller
board_build.mcu = stm32g474ret6

; change MCU frequency
board_build.f_cpu = 170000000L
```
Uploading

Nucleo G474RE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```ini
[env:nucleo_g474re]
platform = stm32
board = nucleo_g474re
upload_protocol = mbed
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

Nucleo G474RE does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

OLIMEXINO-STM32

Contents

- OLIMEXINO-STM32
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Olimex</td>
</tr>
</tbody>
</table>
Configuration

Please use `olimexino` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:olimexino]
platform = ststm32
board = olimexino
```

You can override default OLIMEXINO-STM32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `olimexino.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:olimexino]
platform = ststm32
board = olimexino

; change microcontroller
board_build.mcu = stm32f103rbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

OLIMEXINO-STM32 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:olimexino]
platform = ststm32
board = olimexino

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).
OLIMEXINO-STM32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### Olimex STM32-H103

- **Olimex STM32-H103**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Olimex</td>
</tr>
</tbody>
</table>

Configuration

Please use `olimex_f103` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:olimex_f103]
platform = ststm32
board = olimex_f103
```

You can override default Olimex STM32-H103 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `olimex_f103.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:olimex_f103]
platform = ststm32
board = olimex_f103

; change microcontroller
board_build.mcu = stm32f103rbl6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Olimex STM32-H103 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:olimex_f103]
platform = ststm32
board = olimex_f103
```

(continues on next page)
upload_protocol = stlink

Debugging

_DEBUGging_ - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging _Tools & Debug Probes_ using _debug_tool_ option in “platformio.ini” (Project Configuration File).

Olimex STM32-H103 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Olimex STM32-P405
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F405RGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Olimex</td>
</tr>
</tbody>
</table>

Configuration

Please use `olimex_p405` ID for `board` option in “platformio.ini” (**Project Configuration File**):

```
[env:olimex_p405]
platform = ststm32
board = olimex_p405
```

You can override default Olimex STM32-P405 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `olimex_p405.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:olimex_p405]
platform = ststm32
board = olimex_p405

; change microcontroller
board_build.mcu = stm32f405rgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

Olimex STM32-P405 supports the next uploading protocols:
- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:olimex_p405]
platform = stm32
board = olimex_p405
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Olimex STM32-P405 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

P-Nucleo WB55RG

Contents

- P-Nucleo WB55RG
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32WB55RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192.00KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use `nucleo_wb55rg_p` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:nucleo_wb55rg_p]
platform = ststm32
board = nucleo_wb55rg_p
```

You can override default P-Nucleo WB55RG settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_wb55rg_p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_wb55rg_p]
platform = ststm32
board = nucleo_wb55rg_p

; change microcontroller
board_build.mcu = stm32wb55rg

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

P-Nucleo WB55RG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_wb55rg_p]
platform = ststm32
board = nucleo_wb55rg_p

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

1980 Chapter 1. Contents
P-Nucleo WB55RG has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

PYBSTICK26 Duino

Contents

- PYBSTICK26 Duino
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F072RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PYBStick</td>
</tr>
</tbody>
</table>

Configuration

Please use pybstick26_duino ID for board option in “platformio.ini” (Project Configuration File):
You can override default PYBSTICK26 Duino settings per build environment using board_*** option, where *** is a JSON object path from board manifest pybstick26_duino.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:pybstick26_duino]
platform = ststm32
board = pybstick26_duino

; change microcontroller
board_build.mcu = stm32f072rb

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

PYBSTICK26 Duino supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:pybstick26_duino]
platform = ststm32
board = pybstick26_duino

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

PYBSTICK26 Duino does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
## Compatible Tools

<table>
<thead>
<tr>
<th></th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

## PYBSTick 26 Pro

### Contents

- PYBSTick 26 Pro
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

Configuration

Please use pybstick26_pro ID for board option in "platformio.ini" (Project Configuration File):

```
[env:pybstick26_pro]
platform = ststm32
board = pybstick26_pro
```

You can override default PYBStick 26 Pro settings per build environment using board_*** option, where *** is a JSON object path from board manifest pybstick26_pro.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:pybstick26_pro]
platform = ststm32
board = pybstick26_pro

; change microcontroller
board_build.mcu = stm32f412re

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

PYBStick 26 Pro supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:pybstick26_pro]
platform = ststm32
board = pybstick26_pro

upload_protocol = stlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

PYBStick 26 Pro does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

PYBStick Lite 26

Contents

- PYBStick Lite 26
  - Hardware
  - Configuration
  - Uploading

1.12. Boards
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CEU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PYBStick</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `pybstick26_lite` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:pybstick26_lite]
platform = stm32
board = pybstick26_lite
```

You can override default PYBStick Lite 26 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `pybstick26_lite.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:pybstick26_lite]
platform = stm32
board = pybstick26_lite

; change microcontroller
board_build.mcu = stm32f401ceu6

; change MCU frequency
board_build.f_cpu = 84000000L
```

**Uploading**

PYBStick Lite 26 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink
Default protocol is \texttt{stlink}

You can change upload protocol using \texttt{upload_protocol} option:

```
[env:pybstick26_lite]
platform = ststm32
board = pybstick26_lite
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using \texttt{debug_tool} option in “platformio.ini” (Project Configuration File).

PYBStick Lite 26 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
PYBStick Standard 26

Contents

- PYBStick Standard 26
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411CEU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PYBStick</td>
</tr>
</tbody>
</table>

Configuration

Please use pybstick26_std ID for board option in "platformio.ini" (Project Configuration File):

```
[env:pybstick26_std]
platform = ststm32
board = pybstick26_std
```

You can override default PYBStick Standard 26 settings per build environment using board_*** option, where *** is a JSON object path from board manifest pybstick26_std.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:pybstick26_std]
platform = ststm32
board = pybstick26_std

; change microcontroller
board_build.mcu = stm32f411ceu6

; change MCU frequency
board_build.f_cpu = 100000000L
```
Uploading

PYBStick Standard 26 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:pybstick26_std]
platform = ststm32
board = pybstick26_std
upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

PYBStick Standard 26 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Piconomix PX-HER0

Contents

- Piconomix PX-HER0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L072RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Piconomix</td>
</tr>
</tbody>
</table>

Configuration

Please use `piconomix_px_her0` ID for `board` option in “platformio.ini” *(Project Configuration File)*:
You can override default Piconomix PX-HER0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `piconomix_px_her0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:piconomix_px_her0]
platform = ststm32
board = piconomix_px_her0

; change microcontroller
board_build.mcu = stm32l072rb

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Uploading

Piconomix PX-HER0 supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:piconomix_px_her0]
platform = ststm32
board = piconomix_px_her0

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.

Piconomix PX-HER0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | 
J-LINK | | 
ST-LINK | | 

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

PrntrBoard V2

Contents

- PrntrBoard V2
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>PrntrBoard</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `prntr_v2` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:prntr_v2]
platform = ststm32
board = prntr_v2
```

You can override default PrntrBoard V2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `prntr_v2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:prntr_v2]
platform = ststm32
board = prntr_v2

; change microcontroller
board_build.mcu = stm32f407re

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

PrntrBoard V2 supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:prntr_v2]
platform = ststm32
board = prntr_v2

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

PrntrBoard V2 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EF32 and others</td>
</tr>
</tbody>
</table>

RAK811 LoRa Tracker

Contents

- RAK811 LoRa Tracker
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L151RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RAK</td>
</tr>
</tbody>
</table>

Configuration

Please use rak811_tracker ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:rak811_tracker]
platform = ststm32
board = rak811_tracker
```

You can override default RAK811 LoRa Tracker settings per build environment using `board_***` option, where *** is a JSON object path from board manifest rak811_tracker.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:rak811_tracker]
platform = ststm32
board = rak811_tracker

; change microcontroller
board_build.mcu = stm32l151rbt6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

RAK811 LoRa Tracker supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
[env:rak811_tracker]
platform = stm32
board = rak811_tracker
upload_protocol = stlink

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

RAK811 LoRa Tracker does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

RAK811 LoRa Tracker
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L151RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RAK</td>
</tr>
</tbody>
</table>

Configuration

Please use `rak811_tracker_32` ID for `board` option in "platformio.ini" (Project Configuration File):

```plaintext
[env:rak811_tracker_32]
platform = ststm32
board = rak811_tracker_32
```

You can override default RAK811 LoRa Tracker settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `rak811_tracker_32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:rak811_tracker_32]
platform = ststm32
board = rak811_tracker_32

; change microcontroller
board_build.mcu = stm32l151rbt6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

RAK811 LoRa Tracker supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• serial
• stlink
Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:rak811_tracker_32]
platform = stm32
board = rak811_tracker_32
upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

RAK811 LoRa Tracker does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**RHF76 052**

**Contents**

- **RHF76 052**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L051C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `rhf76_052` ID for board option in “platformio.ini” (Project Configuration File):
You can override default RHF76 052 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `rhf76_052.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:rhf76_052]
platform = ststm32
board = rhf76_052

; change microcontroller
board_build.mcu = stm32l051c8t6

; change MCU frequency
board_build.f_cpu = 32000000
```

### Uploading

RHF76 052 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:rhf76_052]
platform = ststm32
board = rhf76_052

upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

RHF76 052 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Arduino</td>
</tr>
<tr>
<td>CMSIS</td>
</tr>
<tr>
<td>STM32Cube</td>
</tr>
<tr>
<td>libOpenCM3</td>
</tr>
</tbody>
</table>

**RushUp Cloud-JAM**

**Contents**

- *RushUp Cloud-JAM*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RushUp</td>
</tr>
</tbody>
</table>

## Configuration

Please use `cloud_jam` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:cloud_jam]
platform = ststm32
board = cloud_jam
```

You can override default RushUp Cloud-JAM settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cloud_jam.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:cloud_jam]
platform = ststm32
board = cloud_jam

; change microcontroller
board_build.mcu = stm32f401ret6

; change MCU frequency
board_build.f_cpu = 84000000L
```

## Uploading

RushUp Cloud-JAM supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:cloud_jam]
platform = ststm32
board = cloud_jam

upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

RushUp Cloud-JAM has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

RushUp Cloud-JAM L4

Contents

- RushUp Cloud-JAM L4
  - Hardware
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L476RGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RushUp</td>
</tr>
</tbody>
</table>

Configuration

Please use `cloud_jam_l4` ID for `board` option in "`platformio.ini`" (Project Configuration File):

```ini
[env:cloud_jam_l4]
platform = ststm32
board = cloud_jam_l4
```

You can override default RushUp Cloud-JAM L4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `cloud_jam_l4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:cloud_jam_l4]
platform = ststm32
board = cloud_jam_l4

; change microcontroller
board_build.mcu = stm32l476rgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

RushUp Cloud-JAM L4 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
• stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```yaml
[env:cloud_jam_l4]
platform = ststm32
board = cloud_jam_l4
upload_protocol = stlink
```

## Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using debug_tool option in **“platformio.ini” (Project Configuration File)**.

RushUp Cloud-JAM L4 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

### Compatible Tools

<table>
<thead>
<tr>
<th>Tool Code</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST 32F3348DISCOVERY

Contents

- ST 32F3348DISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F334C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>12KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_f334c8 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_f334c8]
platform = ststm32
board = disco_f334c8
```

You can override default ST 32F3348DISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_f334c8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_f334c8]
platform = ststm32
board = disco_f334c8

; change microcontroller
board_build.mcu = stm32f334c8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

ST 32F3348DISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f334c8]
platform = ststm32
board = disco_f334c8
upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST 32F3348DISCOVERY has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST 32F401CDISCOVERY

Contents

- **ST 32F401CDISCOVERY**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401VCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `disco_f401vc` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default ST 32F401CDISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `disco_f401vc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_f401vc]
platform = ststm32
board = disco_f401vc

; change microcontroller
board_build.mcu = stm32f401vct6

; change MCU frequency
board_build.f_cpu = 84000000L
```

### Uploading

ST 32F401CDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_f401vc]
platform = ststm32
board = disco_f401vc

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

ST 32F401CDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
PlatformsIO Documentation, Release 5.0.5a1

Compatible Tools | On-board | Default |
---|---|---|
*Black Magic Probe* | | |
CMSIS-DAP | | |
J-LINK | Yes | Yes |
ST-LINK | Yes | Yes |

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CMSIS</strong></td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td><strong>libopencm3</strong></td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### ST 32F411EDISCOVERY

#### Contents

- ST 32F411EDISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
## Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

### Configuration

Please use disco_f411ve ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_f411ve]
platform = ststm32
board = disco_f411ve
```

You can override default ST 32F411EDISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest disco_f411ve.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_f411ve]
platform = ststm32
board = disco_f411ve

; change microcontroller
board_build.mcu = stm32f411vet6

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

ST 32F411EDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_f411ve]
platform = ststm32
board = disco_f411ve

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

ST 32F411EDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STMI</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST 32F413HDISCOVERY**

**Contents**

- **ST 32F413HDISCOVERY**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F413ZHT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_f413zh ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_f413zh]
platform = ststm32
board = disco_f413zh
```

You can override default ST 32F413HDISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest disco_f413zh.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_f413zh]
platform = ststm32
board = disco_f413zh

; change microcontroller
board_build.mcu = stm32f413zht6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

ST 32F413HDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

ST 32F413HDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tī Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST 32F429IDISCOVERY

Contents

- ST 32F429IDISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F429ZIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_f429zi ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_f429zi]
platform = ststm32
board = disco_f429zi
```

You can override default ST 32F429IDISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_f429zi.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_f429zi]
platform = ststm32
board = disco_f429zi

; change microcontroller
board_build.mcu = stm32f429zit6

; change MCU frequency
board_build.f_cpu = 180000000L
```
Uploading

ST 32F429IDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f429zi]
platform = ststm32
board = disco_f429zi
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST 32F429IDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST 32F469IDISCOVERY

Contents

- ST 32F469IDISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F469NIH6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>384KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use disco_f469ni ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_f469ni]
platform = ststm32
board = disco_f469ni
```

You can override default ST 32F469IDISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_f469ni.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_f469ni]
platform = ststm32
board = disco_f469ni

; change microcontroller
board_build.mcu = stm32f469nih6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

ST 32F469IDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_f469ni]
platform = ststm32
board = disco_f469ni

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).
ST 32F469IDISCOVERY has on-board debug probe and is ready for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST 32F746GDISCOVERY

Contents

- ST 32F746GDISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F746NGH6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `disco_f746ng` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:disco_f746ng]
platform = ststm32
board = disco_f746ng
```

You can override default ST 32F746GDISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `disco_f746ng.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f746ng]
platform = ststm32
board = disco_f746ng

; change microcontroller
board_build.mcu = stm32f746ngh6

; change MCU frequency
board_build.f_cpu = 216000000L
```

**Uploading**

ST 32F746GDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f746ng]
platform = ststm32
board = disco_f746ng

upload_protocol = stlink
```
Debugging

- “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST 32F769IDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CMSIS-DAP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>J-LINK</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ST-LINK</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Tî Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST 32F769IDISCOVERY**

**Contents**

- **ST 32F769IDISCOVERY**
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F769NIH6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_f769ni ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_f769ni]
platform = ststm32
board = disco_f769ni
```

You can override default ST 32F769IDISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest disco_f769ni.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_f769ni]
platform = ststm32
board = disco_f769ni

; change microcontroller
board_build.mcu = stm32f769nih6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

ST 32F769IDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
• mbed
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f769ni]
platform = ststm32
board = disco_f769ni
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST 32F769IDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the 'things' in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST 32L0538DISCOVERY

Contents

- ST 32L0538DISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L053C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use disco_l053c8 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_l053c8]
platform = ststm32
board = disco_l053c8
```

You can override default ST 32L0538DISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_l053c8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_l053c8]
platform = ststm32
board = disco_l053c8

; change microcontroller
board_build.mcu = stm32l053c8t6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST 32L0538DISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_l053c8]
platform = ststm32
board = disco_l053c8

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
ST 32L0538DISCOVERY has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST 32L100DISCOVERY

Contents

- ST 32L100DISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L100RCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use disco_l100rc ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_l100rc]
platform = ststm32
board = disco_l100rc
```

You can override default ST 32L100DISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest disco_l100rc.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_l100rc]
platform = ststm32
board = disco_l100rc

; change microcontroller
board_build.mcu = stm32l100rct6

; change MCU frequency
board_build.f_cpu = 32000000L
```

**Uploading**

ST 32L100DISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_l100rc]
platform = ststm32
board = disco_l100rc

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

1.12. Boards 2027
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ST 32L100DISCOVERY has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tî Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST 32L476GDISCOVERY

Contents

- ST 32L476GDISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L476VGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `disco_l476vg` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:disco_l476vg]
platform = ststm32
board = disco_l476vg
```

You can override default ST 32L476GDISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_l476vg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_l476vg]
platform = ststm32
board = disco_l476vg

; change microcontroller
board_build.mcu = stm32l476vgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ST 32L476GDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_l476vg]
platform = ststm32
board = disco_l476vg

upload_protocol = stlink
```
Debugging

 Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File)*.

ST 32L476GDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the 'things' in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST 32L496GDISCOVERY**

**Contents**

- ST 32L496GDISCOVERY
  - Hardware
  - Configuration
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L496AGI6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `disco_l496ag` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:disco_l496ag]
platform = ststm32
board = disco_l496ag
```

You can override default ST 32L496GDISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest disco_l496ag.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_l496ag]
platform = ststm32
board = disco_l496ag

; change microcontroller
board_build.mcu = stm32l496agi6

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ST 32L496GDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_l496ag]
platform = ststm32
board = disco_l496ag
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

ST 32L496GDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST B-L475E-IOT01A Discovery kit

Contents

• ST B-L475E-IOT01A Discovery kit
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L475VGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_l475vg_iot01a ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_l475vg_iot01a]
platform = ststm32
board = disco_l475vg_iot01a
```

You can override default ST B-L475E-IOT01A Discovery kit settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_l475vg_iot01a.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:disco_l475vg_iot01a]
platform = ststm32
board = disco_l475vg_iot01a

; change microcontroller
board_build.mcu = stm32l475vgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```
Uploading

ST B-L475E-IOT01A Discovery kit supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:disco_l475vg_iot01a]
platform = ststm32
board = disco_l475vg_iot01a
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST B-L475E-IOT01A Discovery kit has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST DISCO-L072CZ-LRWAN1

Contents

- ST DISCO-L072CZ-LRWAN1
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L072CZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
### Configuration

Please use `disco_l072cz_lrwan1` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_l072cz_lrwan1]
platform = ststm32
board = disco_l072cz_lrwan1
```

You can override default ST DISCO-L072CZ-LRWAN1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_l072cz_lrwan1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_l072cz_lrwan1]
platform = ststm32
board = disco_l072cz_lrwan1

; change microcontroller
board_build.mcu = stm32l072cz

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Uploading

ST DISCO-L072CZ-LRWAN1 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_l072cz_lrwan1]
platform = ststm32
board = disco_l072cz_lrwan1

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
ST DISCO-L072CZ-LRWAN1 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST Discovery F072RB**

**Contents**

- **ST Discovery F072RB**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F072RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_f072rb ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:disco_f072rb]
platform = ststm32
board = disco_f072rb
```

You can override default ST Discovery F072RB settings per build environment using `board_***` option, where *** is a JSON object path from board manifest disco_f072rb.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f072rb]
platform = ststm32
board = disco_f072rb

; change microcontroller
board_build.mcu = stm32f072rbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST Discovery F072RB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f072rb]
platform = ststm32
board = disco_f072rb
```

(continues on next page)
upload_protocol = stlink

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Discovery F072RB has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTO</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo F030R8

Contents

- ST Nucleo F030R8
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F030R8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f030r8 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f030r8]
platform = ststm32
board = nucleo_f030r8
```

You can override default ST Nucleo F030R8 settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f030r8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_f030r8]
platform = ststm32
board = nucleo_f030r8

; change microcontroller
board_build.mcu = stm32f030r8t6

; change MCU frequency
board_build.f_cpu = 48000000L
```
**Uploading**

ST Nucleo F030R8 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f030r8]
platform = ststm32
board = nucleo_f030r8
upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F030R8 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F031K6

Contents

- ST Nucleo F031K6
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F031K6T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use nucleo_f031k6 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f031k6]
platform = ststm32
board = nucleo_f031k6
```

You can override default ST Nucleo F031K6 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest nucleo_f031k6.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_f031k6]
platform = ststm32
board = nucleo_f031k6

; change microcontroller
board_build.mcu = stm32f031k6t6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST Nucleo F031K6 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f031k6]
platform = ststm32
board = nucleo_f031k6

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
ST Nucleo F031K6 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM 32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>LibopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST Nucleo F042K6**

**Contents**

- ST Nucleo F042K6
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F042K6T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use nucleo_f042k6 ID for `board` option in “platformio.ini” *(Project Configuration File):*

```
[env:nucleo_f042k6]
platform = ststm32
board = nucleo_f042k6
```

You can override default ST Nucleo F042K6 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_f042k6.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:nucleo_f042k6]
platform = ststm32
board = nucleo_f042k6

; change microcontroller
board_build.mcu = stm32f042k6t6

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

ST Nucleo F042K6 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f042k6]
platform = ststm32
board = nucleo_f042k6

upload_protocol = stlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F042K6 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST Nucleo F070RB

Contents

- ST Nucleo F070RB
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F070RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f070rb ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f070rb]
platform = ststm32
board = nucleo_f070rb
```

You can override default ST Nucleo F070RB settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f070rb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_f070rb]
platform = ststm32
board = nucleo_f070rb

; change microcontroller
board_build.mcu = stm32f070rbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST Nucleo F070RB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is *stlink*

You can change upload protocol using *upload_protocol* option:

```
[env:nucleo_f070rb]
platform = stm32
board = nucleo_f070rb
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (Project Configuration File).

ST Nucleo F070RB has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardised API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOs</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo F072RB

## Contents

- *ST Nucleo F072RB*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

## Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F072RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

## Configuration

Please use nucleo_f072rb ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```
[env:nucleo_f072rb]
platform = ststm32
board = nucleo_f072rb
```

You can override default ST Nucleo F072RB settings per build environment using *board_*** option, where *** is a JSON object path from board manifest nucleo_f072rb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_f072rb]
platform = ststm32
board = nucleo_f072rb

; change microcontroller
board_build.mcu = stm32f072rbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```
Uploading

ST Nucleo F072RB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```python
[env:nucleo_f072rb]
platform = ststm32
board = nucleo_f072rb

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F072RB has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F091RC

Contents

- ST Nucleo F091RC
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F091RCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use nucleo_f091rc ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f091rc]
platform = ststm32
board = nucleo_f091rc
```

You can override default ST Nucleo F091RC settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f091rc.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_f091rc]
platform = ststm32
board = nucleo_f091rc

; change microcontroller
board_build.mcu = stm32f091rct6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST Nucleo F091RC supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:nucleo_f091rc]
platform = ststm32
board = nucleo_f091rc

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).
ST Nucleo F091RC has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### ST Nucleo F103RB

**Contents**

- **ST Nucleo F103RB**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `nucleo_f103rb` ID for `board` option in “platformio.ini” (*Project Configuration File*):

```plaintext
[env:nucleo_f103rb]
platform = ststm32
board = nucleo_f103rb
```

You can override default ST Nucleo F103RB settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_f103rb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:nucleo_f103rb]
platform = ststm32
board = nucleo_f103rb

; change microcontroller
board_build.mcu = stm32f103rbl6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

ST Nucleo F103RB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
[env:nucleo_f103rb]
platform = stm32
board = nucleo_f103rb
upload_protocol = stlink

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ST Nucleo F103RB has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo F207ZG

Contents

- ST Nucleo F207ZG
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F207ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f207zg ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_f207zg]
platform = ststm32
board = nucleo_f207zg
```

You can override default ST Nucleo F207ZG settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f207zg.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_f207zg]
platform = ststm32
board = nucleo_f207zg

; change microcontroller
board_build.mcu = stm32f207zgt6

; change MCU frequency
board_build.f_cpu = 120000000L
```
Uploading

ST Nucleo F207ZG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink.

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f207zg]
platform = ststm32
board = nucleo_f207zg

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F207ZG has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F302R8

Contents

- ST Nucleo F302R8
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F302R8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use `nucleo_f302r8` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:nucleo_f302r8]
platform = ststm32
board = nucleo_f302r8
```

You can override default ST Nucleo F302R8 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f302r8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_f302r8]
platform = ststm32
board = nucleo_f302r8

; change microcontroller
board_build.mcu = stm32f302r8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

ST Nucleo F302R8 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f302r8]
platform = ststm32
board = nucleo_f302r8

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in "platformio.ini" (Project Configuration File).
ST Nucleo F302R8 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### ST Nucleo F303K8

#### Contents

- **ST Nucleo F303K8**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F303K8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>12KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `nucleo_f303k8` ID for *board* option in “platformio.ini” (*Project Configuration File)*:

```ini
[env:nucleo_f303k8]
platform = ststm32
board = nucleo_f303k8
```

You can override default ST Nucleo F303K8 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_f303k8.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:nucleo_f303k8]
platform = ststm32
board = nucleo_f303k8

; change microcontroller
board_build.mcu = stm32f303k8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

ST Nucleo F303K8 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

ST Nucleo F303K8 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo F303RE

Contents

- ST Nucleo F303RE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F303RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f303re ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_f303re]
platform = ststm32
board = nucleo_f303re
```

You can override default ST Nucleo F303RE settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f303re.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_f303re]
platform = ststm32
board = nucleo_f303re

; change microcontroller
board_build.mcu = stm32f303ret6

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

ST Nucleo F303RE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:nucleo_f303re]
platform = ststm32
board = nucleo_f303re

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ST Nucleo F303RE has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F303ZE

Contents

- ST Nucleo F303ZE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F303ZET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use `nucleo_f303ze` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[nucleo_f303ze]
platform = ststm32
board = nucleo_f303ze
```

You can override default ST Nucleo F303ZE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f303ze.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[nucleo_f303ze]
platform = ststm32
board = nucleo_f303ze

; change microcontroller
board_build.mcu = stm32f303zet6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

ST Nucleo F303ZE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[nucleo_f303ze]
platform = ststm32
board = nucleo_f303ze

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
ST Nucleo F303ZE has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

**ST Nucleo F334R8**

**Contents**

- *ST Nucleo F334R8*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F334R8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `nucleo_f334r8` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:nucleo_f334r8]
platform = stm32
board = nucleo_f334r8
```

You can override default ST Nucleo F334R8 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f334r8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_f334r8]
platform = stm32
board = nucleo_f334r8

; change microcontroller
board_build.mcu = stm32f334r8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

ST Nucleo F334R8 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f334r8]
platform = stm32
board = nucleo_f334r8

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ST Nucleo F334R8 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F401RE

Contents

- ST Nucleo F401RE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `nucleo_f401re` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:nucleo_f401re]
platform = ststm32
board = nucleo_f401re
```

You can override default ST Nucleo F401RE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f401re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_f401re]
platform = ststm32
board = nucleo_f401re

; change microcontroller
board_build.mcu = stm32f401ret6

; change MCU frequency
board_build.f_cpu = 84000000L
```

**Uploading**

ST Nucleo F401RE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:nucleo_f401re]
platform = stm32
board = nucleo_f401re
upload_protocol = stlink

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

ST Nucleo F401RE has on-board debug probe and *IS READY* for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CM-SIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+/)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

## ST Nucleo F410RB

### Contents

- **ST Nucleo F410RB**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F410RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

### Configuration

Please use nucleo_f410rb ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_f410rb]
platform = ststm32
board = nucleo_f410rb
```

You can override default ST Nucleo F410RB settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f410rb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_f410rb]
platform = ststm32
board = nucleo_f410rb

; change microcontroller
board_build.mcu = stm32f410rbt6

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

ST Nucleo F410RB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:nucleo_f410rb]
platform = ststm32
board = nucleo_f410rb

upload_protocol = stlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

ST Nucleo F410RB has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F411RE

- **ST Nucleo F411RE**
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f411re ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_f411re]
platform = ststm32
board = nucleo_f411re
```

You can override default ST Nucleo F411RE settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f411re.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_f411re]
platform = ststm32
board = nucleo_f411re

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

ST Nucleo F411RE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f411re]
platform = ststm32
board = nucleo_f411re
upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (*Project Configuration File)*.

ST Nucleo F411RE has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F412ZG

Contents

- ST Nucleo F412ZG
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use nucleo_f412zg ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f412zg]
platform = ststm32
board = nucleo_f412zg
```

You can override default ST Nucleo F412ZG settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f412zg.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_f412zg]
platform = ststm32
board = nucleo_f412zg

; change microcontroller
board_build.mcu = stm32f412zgt6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

ST Nucleo F412ZG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:nucleo_f412zg]
platform = ststm32
board = nucleo_f412zg

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
ST Nucleo F412ZG has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32 Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST Nucleo F413ZH**

**Contents**

- **ST Nucleo F413ZH**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F413ZHT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use nucleo_f413zh ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f413zh]
platform = ststm32
board = nucleo_f413zh
```

You can override default ST Nucleo F413ZH settings per build environment using `board_***` option, where *** is a JSON object path from board manifest nucleo_f413zh.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_f413zh]
platform = ststm32
board = nucleo_f413zh

; change microcontroller
board_build.mcu = stm32f413zht6

; change MCU frequency
board_build.f_cpu = 100000000L
```

**Uploading**

ST Nucleo F413ZH supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f413zh]
platform = ststm32
board = nucleo_f413zh

upload_protocol = stlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F413ZH has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST Nucleo F429ZI

Contents

- **ST Nucleo F429ZI**
  - Hardware
  - Configuration
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F429ZIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use **nucleo_f429zi** ID for **board** option in **"platformio.ini"** *(Project Configuration File)*:

```
[env:nucleo_f429zi]
platform = ststm32
board = nucleo_f429zi
```

You can override default ST Nucleo F429ZI settings per build environment using **board_*** option, where *** is a JSON object path from board manifest **nucleo_f429zi.json**. For example, **board_build.mcu**, **board_build.f_cpu**, etc.

```
[env:nucleo_f429zi]
platform = ststm32
board = nucleo_f429zi

; change microcontroller
board_build.mcu = stm32f429zit6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

ST Nucleo F429ZI supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f429zi]
platform = ststm32
board = nucleo_f429zi

upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F429ZI has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST Nucleo F439ZI

Contents

- **ST Nucleo F439ZI**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F439ZIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use nucleo_f439zi ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_f439zi]
platform = ststm32
board = nucleo_f439zi
```

You can override default ST Nucleo F439ZI settings per build environment using `board_***` option, where *** is a JSON object path from board manifest nucleo_f439zi.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_f439zi]
platform = ststm32
board = nucleo_f439zi

; change microcontroller
board_build.mcu = stm32f439zit6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

ST Nucleo F439ZI supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f439zi]
platform = ststm32
board = nucleo_f439zi

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).
ST Nucleo F439ZI has on-board debug probe and is ready for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST Nucleo F446RE

Contents

- **ST Nucleo F446RE**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F446RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

### Configuration

Please use nucleo_f446re ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f446re]
platform = ststm32
board = nucleo_f446re
```

You can override default ST Nucleo F446RE settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f446re.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_f446re]
platform = ststm32
board = nucleo_f446re

; change microcontroller
board_build.mcu = stm32f446ret6

; change MCU frequency
board_build.f_cpu = 180000000L
```

### Uploading

ST Nucleo F446RE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:nucleo_f446re]
platform = ststm32
board = nucleo_f446re

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F446RE has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F446ZE
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F446ZET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `nucleo_f446ze` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:nucleo_f446ze]
platform = ststm32
board = nucleo_f446ze
```

You can override default ST Nucleo F446ZE settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_f446ze.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_f446ze]
platform = ststm32
board = nucleo_f446ze

; change microcontroller
board_build.mcu = stm32f446zet6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

ST Nucleo F446ZE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f446ze]
platform = ststm32
board = nucleo_f446ze
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F446ZE has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo F722ZE

Contents

- ST Nucleo F722ZE
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F722ZET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f722ze ID for board option in "platformio.ini" (Project Configuration File):

```
[env:nucleo_f722ze]
platform = ststm32
board = nucleo_f722ze
```

You can override default ST Nucleo F722ZE settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f722ze.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_f722ze]
platform = ststm32
board = nucleo_f722ze

; change microcontroller
board_build.mcu = stm32f722zet6

; change MCU frequency
board_build.f_cpu = 216000000L
```
Uploading

ST Nucleo F722ZE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink.

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f722ze]
platform = ststm32
board = nucleo_f722ze
upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F722ZE has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST Nucleo F746ZG

Contents

- ST Nucleo F746ZG
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F746ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f746zg ID for board option in “platformio.ini” (Project Configuration File):
You can override default ST Nucleo F746ZG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f746zg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_f746zg]
platform = ststm32
board = nucleo_f746zg

; change microcontroller
board_build.mcu = stm32f746zgt6

; change MCU frequency
board_build.f_cpu = 216000000
```

### Uploading

ST Nucleo F746ZG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f746zg]
platform = ststm32
board = nucleo_f746zg

upload_protocol = stlink
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F746ZG has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | Yes
CMSIS-DAP | Yes | Yes
J-LINK | | |
ST-LINK | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F756ZG

Contents

* ST Nucleo F756ZG
  * Hardware
  * Configuration
  * Uploading
  * Debugging
  * Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F756ZG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `nucleo_f756zg` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_f756zg]
platform = ststm32
board = nucleo_f756zg
```

You can override default ST Nucleo F756ZG settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_f756zg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_f756zg]
platform = ststm32
board = nucleo_f756zg

; change microcontroller
board_build.mcu = stm32f756zg

; change MCU frequency
board_build.f_cpu = 216000000L
```

**Uploading**

ST Nucleo F756ZG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f756zg]
platform = ststm32
board = nucleo_f756zg

upload_protocol = stlink
```
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (**Project Configuration File**).

ST Nucleo F756ZG has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo F767ZI

Contents

- ST Nucleo F767ZI
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F767ZIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_f767zi ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_f767zi]
platform = ststm32
board = nucleo_f767zi
```

You can override default ST Nucleo F767ZI settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_f767zi.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_f767zi]
platform = ststm32
board = nucleo_f767zi

; change microcontroller
board_build.mcu = stm32f767zit6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

ST Nucleo F767ZI supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
• mbed
• stlink

Default protocol is stlink
You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f767zi]
platform = ststm32
board = nucleo_f767zi
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo F767ZI has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo H743ZI

Contents

- ST Nucleo H743ZI
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>512KB</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use `nucleo_h743zi` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_h743zi]
platform = ststm32
board = nucleo_h743zi
```

You can override default ST Nucleo H743ZI settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_h743zi.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:nucleo_h743zi]
platform = ststm32
board = nucleo_h743zi

; change microcontroller
board_build.mcu = stm32h743zit6

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

ST Nucleo H743ZI supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_h743zi]
platform = ststm32
board = nucleo_h743zi

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
ST Nucleo H743ZI has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STMicroelectronics embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTO</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

**ST Nucleo H745ZI-Q**

**Contents**

- *ST Nucleo H745ZI-Q*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32H745ZIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>480MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `nucleo_h745zi_q` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:nucleo_h745zi_q]
platform = ststm32
board = nucleo_h745zi_q
```

You can override default ST Nucleo H745ZI-Q settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_h745zi_q.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_h745zi_q]
platform = ststm32
board = nucleo_h745zi_q

; change microcontroller
board_build.mcu = stm32h745zit6

; change MCU frequency
board_build.f_cpu = 480000000L
```

**Uploading**

ST Nucleo H745ZI-Q supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_h745zi_q]
platform = ststm32
board = nucleo_h745zi_q

upload_protocol = stlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo H745ZI-Q has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

ST Nucleo L011K4

Contents

- **ST Nucleo L011K4**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L011K4T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l011k4 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_l011k4]
platform = ststm32
board = nucleo_l011k4
```

You can override default ST Nucleo L011K4 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_l011k4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_l011k4]
platform = ststm32
board = nucleo_l011k4

; change microcontroller
board_build.mcu = stm32l011k4t6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST Nucleo L011K4 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l011k4]
platform = ststm32
board = nucleo_l011k4
```

(continues on next page)
upload_protocol = stlink

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

ST Nucleo L011K4 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST Nucleo L031K6**

Contents

- **ST Nucleo L031K6**
  - Hardware
  - Configuration
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L031K6T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l031k6 ID for *board* option in “platformio.ini” *(Project Configuration File):*

```
[env:nucleo_l031k6]
platform = ststm32
board = nucleo_l031k6
```

You can override default ST Nucleo L031K6 settings per build environment using *board_*** option, where *** is a JSON object path from board manifest nucleo_l031k6.json. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:nucleo_l031k6]
platform = ststm32
board = nucleo_l031k6

; change microcontroller
board_build.mcu = stm32l031k6t6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST Nucleo L031K6 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l031k6]
platform = stm32
board = nucleo_l031k6
upload_protocol = stlink
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in **“platformio.ini” (Project Configuration File)**.

ST Nucleo L031K6 has on-board debug probe and **is ready** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>CMSIS</strong></td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td><strong>libopenCM3</strong></td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Tý Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo L053R8

Contents

- ST Nucleo L053R8
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L053R8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l053r8 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_l053r8]
platform = stm32
board = nucleo_l053r8
```

You can override default ST Nucleo L053R8 settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_l053r8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_l053r8]
platform = stm32
board = nucleo_l053r8

; change microcontroller
board_build.mcu = stm32l053r8t6

; change MCU frequency
board_build.f_cpu = 32000000L
```
Uploading

ST Nucleo L053R8 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_l053r8]
platform = ststm32
board = nucleo_1053r8
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo L053R8 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo L073RZ

Contents

- ST Nucleo L073RZ
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L073RZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

1.12. Boards
Configuration

Please use nucleo_1073rz ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_1073rz]
platform = ststm32
board = nucleo_1073rz
```

You can override default ST Nucleo L073RZ settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_1073rz.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_1073rz]
platform = ststm32
board = nucleo_1073rz

; change microcontroller
board_build.mcu = stm32l073rz

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST Nucleo L073RZ supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:nucleo_1073rz]
platform = ststm32
board = nucleo_1073rz

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
ST Nucleo L073RZ has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### ST Nucleo L152RE

**Contents**

- **ST Nucleo L152RE**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L152RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>80KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `nucleo_l152re` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_l152re]
platform = ststm32
board = nucleo_l152re
```

You can override default ST Nucleo L152RE settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `nucleo_l152re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_l152re]
platform = ststm32
board = nucleo_l152re

; change microcontroller
board_build.mcu = stm32l152ret6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST Nucleo L152RE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo L152RE has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, T Freescale and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo L412KB

Contents

- ST Nucleo L412KB
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L412KBU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>40KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l412kb ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_l412kb]
platform = ststm32
board = nucleo_l412kb
```

You can override default ST Nucleo L412KB settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_l412kb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_l412kb]
platform = ststm32
board = nucleo_l412kb

; change microcontroller
board_build.mcu = stm32l412kbu6

; change MCU frequency
board_build.f_cpu = 80000000L
```
**Uploading**

ST Nucleo L412KB supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_l412kb]
platform = ststm32
board = nucleo_l412kb
upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo L412KB has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
</tbody>
</table>

ST Nucleo L432KC

Contents

- ST Nucleo L432KC
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L432KCU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l432kc ID for board option in “platformio.ini” (Project Configuration File):
You can override default ST Nucleo L432KC settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_l432kc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_l432kc]
platform = ststm32
board = nucleo_l432kc

; change microcontroller
board_build.mcu = stm32l432kcu6

; change MCU frequency
board_build.f_cpu = 80000000L
```

## Uploading

ST Nucleo L432KC supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l432kc]
platform = ststm32
board = nucleo_l432kc

upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

⚠️ **Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

ST Nucleo L432KC has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | Yes
CMSIS-DAP | Yes | Yes
J-LINK | Yes | Yes
ST-LINK | Yes | Yes

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

## ST Nucleo L433RC-P

### Contents

- **ST Nucleo L433RC-P**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L433RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use nucleo_l433rc_p ID for `board` option in **“platformio.ini” (Project Configuration File):**

```
[env:nucleo_l433rc_p]
platform = ststm32
board = nucleo_l433rc_p
```

You can override default ST Nucleo L433RC-P settings per build environment using `board_***` option, where *** is a JSON object path from board manifest nucleo_l433rc_p.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_l433rc_p]
platform = ststm32
board = nucleo_l433rc_p

; change microcontroller
board_build.mcu = stm32l433rc

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ST Nucleo L433RC-P supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_l433rc_p]
platform = ststm32
board = nucleo_l433rc_p

upload_protocol = stlink
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

ST Nucleo L433RC-P has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST Nucleo L452RE**

**Contents**

- **ST Nucleo L452RE**
  - Hardware
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L452RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use **nucleo_l452re** ID for `board` option in `platformio.ini` (Project Configuration File):

```
[env:nucleo_l452re]
platform = ststm32
board = nucleo_l452re
```

You can override default ST Nucleo L452RE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_l452re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_l452re]
platform = ststm32
board = nucleo_l452re

; change microcontroller
board_build.mcu = stm32l452ret6

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ST Nucleo L452RE supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_l452re]
platform = ststm32
board = nucleo_l452re
upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

ST Nucleo L452RE has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td><strong>Yes</strong></td>
<td><strong>Yes</strong></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo L476RG

Contents

• ST Nucleo L476RG
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L476RGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l476rg ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_l476rg]
platform = ststm32
board = nucleo_l476rg
```

You can override default ST Nucleo L476RG settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_l476rg.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_l476rg]
platform = ststm32
board = nucleo_l476rg

; change microcontroller
board_build.mcu = stm32l476rgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```
Uploading

ST Nucleo L476RG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_l476rg]
platform = ststm32
board = nucleo_l476rg
upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo L476RG has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>libopencm3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EF3M32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo L486RG

Contents

- ST Nucleo L486RG
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L486RGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use `nucleo_l486rg` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_l486rg]
platform = ststm32
board = nucleo_l486rg
```

You can override default ST Nucleo L486RG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_l486rg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_l486rg]
platform = ststm32
board = nucleo_l486rg

; change microcontroller
board_build.mcu = stm32l486rgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ST Nucleo L486RG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_l486rg]
platform = ststm32
board = nucleo_l486rg

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
ST Nucleo L486RG has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### ST Nucleo L496ZG

#### Contents

- **ST Nucleo L496ZG**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L496ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

## Configuration

Please use `nucleo_l496zg` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:nucleo_l496zg]
platform = ststm32
board = nucleo_l496zg
```

You can override default ST Nucleo L496ZG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_l496zg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:nucleo_l496zg]
platform = ststm32
board = nucleo_l496zg

; change microcontroller
board_build.mcu = stm32l496zgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```

## Uploading

ST Nucleo L496ZG supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l496zg]
platform = ststm32
board = nucleo_l496zg

upload_protocol = stlink
```

## Debugging

*Debugging* - "1-click" solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

ST Nucleo L496ZG has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST Nucleo L496ZG-P

Contents

- ST Nucleo L496ZG-P
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L496ZGT6P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l496zg_p ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_l496zg_p]
platform = ststm32
board = nucleo_l496zg_p
```

You can override default ST Nucleo L496ZG-P settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_l496zg_p.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_l496zg_p]
platform = ststm32
board = nucleo_l496zg_p

; change microcontroller
board_build.mcu = stm32l496zgt6p

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ST Nucleo L496ZG-P supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_1496zg_p]
platform = ststm32
board = nucleo_1496zg_p
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

ST Nucleo L496ZG-P has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Tī Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
ST Nucleo L4R5ZI

Contents

- ST Nucleo L4R5ZI
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L4R5ZIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>640KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_l4r5zi ID for board option in “platformio.ini” (Project Configuration File):

```
[env:nucleo_l4r5zi]
platform = ststm32
board = nucleo_l4r5zi
```

You can override default ST Nucleo L4R5ZI settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_l4r5zi.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:nucleo_l4r5zi]
platform = ststm32
board = nucleo_l4r5zi

; change microcontroller
board_build.mcu = stm32l4r5zit6

; change MCU frequency
board_build.f_cpu = 120000000L
```
Uploading

ST Nucleo L4R5ZI supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is **stlink**

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:nucleo_l4r5zi]
platform = ststm32
board = nucleo_l4r5zi
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Nucleo L4R5ZI has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

ST STM32F0308DISCOVERY

Contents

- **ST STM32F0308DISCOVERY**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F030R8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use `disco_f030r8` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:disco_f030r8]
platform = ststm32
board = disco_f030r8
```

You can override default ST STM32F0308DISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `disco_f030r8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f030r8]
platform = ststm32
board = disco_f030r8

; change microcontroller
board_build.mcu = stm32f030r8t6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST STM32F0308DISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f030r8]
platform = ststm32
board = disco_f030r8

upload_protocol = stlink
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in "platformio.ini" (Project Configuration File).
ST STM32F0308DISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### ST STM32F0DISCOVERY

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F051R8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use disco_f051r8 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_f051r8]
platform = ststm32
board = disco_f051r8
```

You can override default ST STM32F0DISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_f051r8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_f051r8]
platform = ststm32
board = disco_f051r8

; change microcontroller
board_build.mcu = stm32f051r8t6

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Uploading**

ST STM32F0DISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_f051r8]
platform = ststm32
board = disco_f051r8

upload_protocol = stlink
```
Debugging

- “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST STM32F0DISCOVERY has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

ST STM32F3DISCOVERY

**Contents**

- ST STM32F3DISCOVERY
  - Hardware
  - Configuration
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F303VCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>48KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `disco_f303vc` ID for **board** option in “platformio.ini” *(Project Configuration File):*

```ini
[env:disco_f303vc]
platform = ststm32
board = disco_f303vc
```

You can override default ST STM32F3DISCOVERY settings per build environment using **board_*** option, where *** is a JSON object path from board manifest `disco_f303vc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_f303vc]
platform = ststm32
board = disco_f303vc

; change microcontroller
board_build.mcu = stm32f303vct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

ST STM32F3DISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink
Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f303vc]
platform = stm32
board = disco_f303vc
upload_protocol = stlink
```

---

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST STM32F3DISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST STM32F4DISCOVERY

Contents

- ST STM32F4DISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
## Configuration

Please use `disco_f407vg` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:disco_f407vg]
platform = ststm32
board = disco_f407vg
```

You can override default ST STM32F4DISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_f407vg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:disco_f407vg]
platform = ststm32
board = disco_f407vg

; change microcontroller
board_build.mcu = stm32f407vgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

## Uploading

ST STM32F4DISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_f407vg]
platform = ststm32
board = disco_f407vg

upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST STM32F4DISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST STM32G0316-DISCO**
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32G031J6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_g031j6 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_g031j6]
platform = ststm32
board = disco_g031j6
```

You can override default ST STM32G0316-DISCO settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_g031j6.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_g031j6]
platform = ststm32
board = disco_g031j6

; change microcontroller
board_build.mcu = stm32g031j6

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

ST STM32G0316-DISCO supports the next uploading protocols:
• blackmagic
• cmsis-dap
• dfu
• jlink
• mbed
• serial

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```ini
[env:disco_g031j6]
platform = ststm32
board = disco_g031j6
upload_protocol = mbed
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST STM32G0316-DISCO does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ST STM32L073Z-EVAL

Contents

- **ST STM32L073Z-EVAL**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L073VZT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `eval_l073z` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default ST STM32L073Z-EVAL settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `eval_l073z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:eval_l073z]
platform = ststm32
board = eval_l073z

; change microcontroller
board_build.mcu = stm32l073vzt6

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Uploading

ST STM32L073Z-EVAL supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:eval_l073z]
platform = ststm32
board = eval_l073z

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “`platformio.ini` (Project Configuration File).”

ST STM32L073Z-EVAL has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### ST STM32LDISCOVERY

#### Contents

- **ST STM32LDISCOVERY**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L152RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use disco_l152rb ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_l152rb]
platform = ststm32
board = disco_l152rb
```

You can override default ST STM32LDISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_l152rb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:disco_l152rb]
platform = ststm32
board = disco_l152rb

; change microcontroller
board_build.mcu = stm32l152rbt6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST STM32LDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:disco_l152rb]
platform = ststm32
board = disco_l152rb

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).
ST STM32LDISCOVERY has on-board debug probe and is ready for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CMSIS</strong></td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td><strong>ST Standard Peripheral Library</strong></td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td><strong>libOpenCM3</strong></td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST STM32VLDISCOVERY**

- **ST STM32VLDISCOVERY**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F100RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>24MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_f100rb ID for board option in “platformio.ini” (Project Configuration File):

```
[env:disco_f100rb]
platform = ststm32
board = disco_f100rb
```

You can override default ST STM32VLDISCOVERY settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_f100rb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:disco_f100rb]
platform = ststm32
board = disco_f100rb

; change microcontroller
board_build.mcu = stm32f100rbt6

; change MCU frequency
board_build.f_cpu = 24000000L
```

Uploading

ST STM32VLDISCOVERY supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_f100rb]
platform = ststm32
board = disco_f100rb
```

(continues on next page)
upload_protocol = stlink

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST STM32VLDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ST Sensor Node**
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L476JG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Avnet Silica</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `silica_sensor_node` ID for `board` option in “`platformio.ini` (Project Configuration File):`

```
[env:silica_sensor_node]
platform = ststm32
board = silica_sensor_node

```

You can override default ST Sensor Node settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `silica_sensor_node.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:silica_sensor_node]
platform = ststm32
board = silica_sensor_node

; change microcontroller
board_build.mcu = stm32l476jg

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

ST Sensor Node supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• mbed
• stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:silica_sensor_node]
platform = ststm32
board = silica_sensor_node
upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

ST Sensor Node has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CMSIS-DAP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>J-LINK</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ST-LINK</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STEVAL-FCU001V1 Flight controller unit evaluation board

Contents

- STEVAL-FCU001V1 Flight controller unit evaluation board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CCU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use steval_fcu001v1 ID for board option in “platformio.ini” (Project Configuration File):
[env:steval_fcu001v1]
platform = ststm32
board = steval_fcu001v1

You can override default STEVAL-FCU001V1 Flight controller unit evaluation board settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `steval_fcu001v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

[env:steval_fcu001v1]
platform = ststm32
board = steval_fcu001v1

; change microcontroller
board_build.mcu = stm32f401ccu6

; change MCU frequency
board_build.f_cpu = 84000000L

**Uploading**

STEVAL-FCU001V1 Flight controller unit evaluation board supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

[env:steval_fcu001v1]
platform = ststm32
board = steval_fcu001v1

upload_protocol = stlink

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

STEVAL-FCU001V1 Flight controller unit evaluation board does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tī Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM32-E407

#### Contents

- **STM32-E407**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Olimex</td>
</tr>
</tbody>
</table>

## Configuration

Please use `olimex_e407` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:olimex_e407]
platform = ststm32
board = olimex_e407
```

You can override default STM32-E407 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `olimex_e407.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:olimex_e407]
platform = ststm32
board = olimex_e407

; change microcontroller
board_build.mcu = stm32f407zgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

## Uploading

STM32-E407 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:olimex_e407]
platform = ststm32
board = olimex_e407

upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

STM32-E407 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM32-H407

**Contents**

- *STM32-H407*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *ST STM32*: The ST M32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407ZGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Olimex</td>
</tr>
</tbody>
</table>

Configuration

Please use `olimex_h407` ID for `board` option in "platformio.ini" *(Project Configuration File):

```
[env:olimex_h407]
platform = ststm32
board = olimex_h407
```

You can override default STM32-H407 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `olimex_h407.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:olimex_h407]
platform = ststm32
board = olimex_h407

; change microcontroller
board_build.mcu = stm32f407zgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

STM32-H407 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:olimex_h407]
platform = ststm32
board = olimex_h407
```

(continues on next page)
upload_protocol = stlink

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File*).

STM32-H407 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM3210C-EVAL
PlatformIO Documentation, Release 5.0.5a1

Contents

- **STM3210C-EVAL**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F107VCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `eval_f107vc` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:eval_f107vc]
platform = ststm32
board = eval_f107vc
```

You can override default STM3210C-EVAL settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `eval_f107vc.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:eval_f107vc]
platform = ststm32
board = eval_f107vc

; change microcontroller
board_build.mcu = stm32f107vct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM3210C-EVAL supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:eval_f107vc]
platform = ststm32
board = eval_f107vc

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM3210C-EVAL does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

STM32373C-EVAL

Contents

- STM32373C-EVAL
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F373VCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use eval_f373vc ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32373C-EVAL settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `eval_f373vc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:eval_f373vc]
platform = ststm32
board = eval_f373vc

; change microcontroller
board_build.mcu = stm32f373vct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32373C-EVAL supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:eval_f373vc]
platform = ststm32
board = eval_f373vc

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” *(Project Configuration File)*.

STM32373C-EVAL does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tī Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

**STM32F072-EVAL**

**Contents**

- STM32F072-EVAL
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
Configuration

Please use *eval_f072vb* ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:eval_f072vb]
platform = ststm32
board = eval_f072vb
```

You can override default STM32F072-EVAL settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `eval_f072vb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.:

```
[env:eval_f072vb]
platform = ststm32
board = eval_f072vb

; change microcontroller
board_build.mcu = stm32f072vbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

STM32F072-EVAL supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is *stlink*

You can change upload protocol using `upload_protocol` option:

```
[env:eval_f072vb]
platform = ststm32
board = eval_f072vb

upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

STM32F072-EVAL does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM32F103C4 (6k RAM. 16k Flash)
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103C4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:genericSTM32F103C4]
platform = ststm32
board = genericSTM32F103C4
```

You can override default STM32F103C4 (6k RAM. 16k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103C4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103C4]
platform = ststm32
board = genericSTM32F103C4

; change microcontroller
board_build.mcu = stm32f103c4

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103C4 (6k RAM. 16k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

STM32F103C4 (6k RAM. 16k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103C6 (10k RAM. 32k Flash)
PlatformIO Documentation, Release 5.0.5a1

- **STM32F103C6 (10k RAM. 32k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F103C6` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:genericSTM32F103C6]
platform = ststm32
board = genericSTM32F103C6
```

You can override default STM32F103C6 (10k RAM. 32k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103C6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103C6]
platform = ststm32
board = genericSTM32F103C6

; change microcontroller
board_build.mcu = stm32f103c6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103C6 (10k RAM. 32k Flash) supports the next uploading protocols:

- blackmagic
- dfu

1.12. Boards
Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103C6]
platform = ststm32
board = genericSTM32F103C6
upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “`platformio.ini` (Project Configuration File).”

STM32F103C6 (10k RAM. 32k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F103C8 (20k RAM. 64k Flash)

Contents

- STM32F103C8 (20k RAM. 64k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103C8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103C8` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:genericSTM32F103C8]
platform = ststm32
board = genericSTM32F103C8
```

You can override default STM32F103C8 (20k RAM. 64k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103C8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103C8]
platform = ststm32
board = genericSTM32F103C8

; change microcontroller
board_build.mcu = stm32f103c8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

STM32F103C8 (20k RAM, 64k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103C8]
platform = ststm32
board = genericSTM32F103C8
upload_protocol = stlink
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103C8 (20k RAM, 64k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103CB (20k RAM. 128k Flash)

Contents

- STM32F103CB (20k RAM. 128k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>
**Configuration**

Please use `genericSTM32F103CB` ID for `board` option in "`platformio.ini` (Project Configuration File):

```
[env:genericSTM32F103CB]
platform = ststm32
board = genericSTM32F103CB
```

You can override default STM32F103CB (20k RAM, 128k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103CB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103CB]
platform = ststm32
board = genericSTM32F103CB

; change microcontroller
board_build.mcu = stm32f103cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103CB (20k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103CB]
platform = ststm32
board = genericSTM32F103CB

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103CB (20k RAM. 128k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103R4 (6k RAM. 16k Flash)

Contents

- STM32F103R4 (6k RAM. 16k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F103R4` ID for `board` option in "`platformio.ini` (Project Configuration File):

```ini
[env:genericSTM32F103R4]
platform = ststm32
board = genericSTM32F103R4
```

You can override default STM32F103R4 (6k RAM, 16k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103R4.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103R4]
platform = ststm32
board = genericSTM32F103R4

; change microcontroller
board_build.mcu = stm32f103r4

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103R4 (6k RAM, 16k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103R4]
platform = ststm32
board = genericSTM32F103R4

upload_protocol = serial
```
Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

STM32F103R4 (6k RAM, 16k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103R6 (10k RAM, 32k Flash)

Contents

- **STM32F103R6 (10k RAM, 32k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103R6` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:genericSTM32F103R6]
platform = ststm32
board = genericSTM32F103R6
```

You can override default STM32F103R6 (10k RAM. 32k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103R6.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:genericSTM32F103R6]
platform = ststm32
board = genericSTM32F103R6

; change microcontroller
board_build.mcu = stm32f103r6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103R6 (10k RAM. 32k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103R6 (10k RAM. 32k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103R8 (20k RAM. 64 Flash)
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ST32F103R8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103R8` ID for `board` option in `platformio.ini` *(Project Configuration File)*:

```ini
[env:genericSTM32F103R8]
platform = ststm32
board = genericSTM32F103R8
```

You can override default ST32F103R8 (20k RAM. 64 Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103R8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103R8]
platform = ststm32
board = genericSTM32F103R8

; change microcontroller
board_build.mcu = st32f103r8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103R8 (20k RAM. 64 Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
• dfu
• jlink
• serial
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103R8]
platform = ststm32
board = genericSTM32F103R8
upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in `platformio.ini` (Project Configuration File).

STM32F103R8 (20k RAM, 64 Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103RB (20k RAM. 128k Flash)

Contents

- STM32F103RB (20k RAM. 128k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103RB ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F103RB (20k RAM, 128k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103RB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103RB]
platform = ststm32
board = genericSTM32F103RB

; change microcontroller
board_build.mcu = stm32f103rbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32F103RB (20k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103RB]
platform = ststm32
board = genericSTM32F103RB

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
STM32F103RB (20k RAM. 128k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F103RC (48k RAM. 256k Flash)**

**Contents**

- STM32F103RC (48k RAM. 256k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-
time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>48KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F103RC` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:genericSTM32F103RC]
platform = ststm32
board = genericSTM32F103RC
```

You can override default STM32F103RC (48k RAM, 256k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103RC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103RC]
platform = ststm32
board = genericSTM32F103RC

; change microcontroller
board_build.mcu = stm32f103rct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103RC (48k RAM, 256k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103RC]
platform = ststm32
board = genericSTM32F103RC

upload_protocol = stlink
```
Debugging

- “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

STM32F103RC (48k RAM, 256k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, TI Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103RD (64k RAM. 384k Flash)

Contents

- STM32F103RD (64k RAM. 384k Flash)
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>384KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103RD` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:genericSTM32F103RD]
platform = ststm32
board = genericSTM32F103RD
```

You can override default STM32F103RD (64k RAM, 384k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103RD.json`. For example,

```
[env:genericSTM32F103RD]
platform = ststm32
board = genericSTM32F103RD

; change microcontroller
board_build.mcu = stm32f103rd

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103RD (64k RAM, 384k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink
PlatformIO Documentation, Release 5.0.5a1

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103RD]
platform = ststm32
board = genericSTM32F103RD

upload_protocol = serial
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103RD (64k RAM, 384k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103RE (64k RAM, 512k Flash)
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103RE` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F103RE]
platform = ststm32
board = genericSTM32F103RE
```

You can override default STM32F103RE (64k RAM. 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103RE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103RE]
platform = ststm32
board = genericSTM32F103RE

; change microcontroller
board_build.mcu = stm32f103ret6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103RE (64k RAM. 512k Flash) supports the next uploading protocols:
• blackmagic
• cmsis-dap
• dfu
• jlink
• serial
• stlink

Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103RE]
platform = ststm32
board = genericSTM32F103RE
upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103RE (64k RAM. 512k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103RF (96k RAM, 768k Flash)

Contents

- STM32F103RF (96k RAM, 768k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>768KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103RF ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F103RF (96k RAM. 768k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103RF.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103RF]
platform = ststm32
board = genericSTM32F103RF

; change microcontroller
board_build.mcu = stm32f103rf

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32F103RF (96k RAM. 768k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103RF]
platform = ststm32
board = genericSTM32F103RF

upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103RF (96k RAM. 768k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Platforms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>On-board</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### STM32F103RG (96k RAM. 1024k Flash)

#### Contents

- **STM32F103RG (96k RAM. 1024k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
**Configuration**

Please use `genericSTM32F103RG` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F103RG]
platform = ststm32
board = genericSTM32F103RG
```

You can override default STM32F103RG (96k RAM, 1024k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103RG.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103RG]
platform = ststm32
board = genericSTM32F103RG

; change microcontroller
board_build.mcu = stm32f103rg

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103RG (96k RAM, 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103RG]
platform = ststm32
board = genericSTM32F103RG

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103RG (96k RAM, 1024k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103T4 (6k RAM, 16k Flash)

Contents

- STM32F103T4 (6k RAM, 16k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
PlatformIO Documentation, Release 5.0.5a1

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103T4` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:genericSTM32F103T4]
platform = ststm32
board = genericSTM32F103T4
```

You can override default STM32F103T4 (6k RAM, 16k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103T4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103T4]
platform = ststm32
board = genericSTM32F103T4

; change microcontroller
board_build.mcu = stm32f103t4

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103T4 (6k RAM, 16k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F103T4 (6k RAM. 16k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103T6 (10k RAM. 32k Flash)

Contents

1.12. Boards
Hardware

Platform ST STM32: The ST32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>10KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103T6 ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F103T6]
platform = ststm32
board = genericSTM32F103T6
```

You can override default STM32F103T6 (10k RAM. 32k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F103T6.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:genericSTM32F103T6]
platform = ststm32
board = genericSTM32F103T6

; change microcontroller
board_build.mcu = stm32f103t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103T6 (10k RAM. 32k Flash) supports the next uploading protocols:

- `blackmagic`
- `dfu`
• jlink
• serial
• stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103T6]
platform = ststm32
board = genericSTM32F103T6
upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103T6 (10k RAM. 32k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F103T8 (20k RAM. 64k Flash)

Contents

- STM32F103T8 (20k RAM. 64k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103T8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103T8` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:genericSTM32F103T8]
platform = ststm32
board = genericSTM32F103T8
```

You can override default STM32F103T8 (20k RAM. 64k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103T8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103T8]
platform = ststm32
board = genericSTM32F103T8

; change microcontroller
board_build.mcu = stm32f103t8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

STM32F103T8 (20k RAM. 64k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103T8]
platform = ststm32
board = genericSTM32F103T8

upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103T8 (20k RAM. 64k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103TB (20k RAM. 128k Flash)

Contents

- STM32F103TB (20k RAM. 128k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103TBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103TB ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F103TB (20k RAM, 128k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103TB.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```yaml
[env:genericSTM32F103TB]
platform = ststm32
board = genericSTM32F103TB

; change microcontroller
board_build.mcu = stm32f103tbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32F103TB (20k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:genericSTM32F103TB]
platform = ststm32
board = genericSTM32F103TB

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File)*.
STM32F103TB (20k RAM, 128k Flash) does not have on-board debug probe and is not ready for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103V8 (20k RAM, 64k Flash)

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `genericSTM32F103V8` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:genericSTM32F103V8]
platform = ststm32
board = genericSTM32F103V8
```

You can override default STM32F103V8 (20k RAM, 64k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103V8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103V8]
platform = ststm32
board = genericSTM32F103V8

; change microcontroller
board_build.mcu = stm32f103v8

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32F103V8 (20k RAM, 64k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103V8]
platform = ststm32
board = genericSTM32F103V8

upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F103V8 (20k RAM. 64k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103VB (20k RAM. 128k Flash)

Contents

- STM32F103VB (20k RAM. 128k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103VB6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103VB ID for board option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F103VB]
platform = stm32
board = genericSTM32F103VB
```

You can override default STM32F103VB (20k RAM, 128k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F103VB.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:genericSTM32F103VB]
platform = stm32
board = genericSTM32F103VB

; change microcontroller
board_build.mcu = stm32f103vbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103VB (20k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:
[env:genericSTM32F103VB]
platform = stm32
board = genericSTM32F103VB
upload_protocol = stlink

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F103VB (20k RAM. 128k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103VC (48k RAM. 256k Flash)
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103VCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>48KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103VC` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:genericSTM32F103VC]
platform = ststm32
board = genericSTM32F103VC
```

You can override default STM32F103VC (48k RAM. 256k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103VC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103VC]
platform = ststm32
board = genericSTM32F103VC

; change microcontroller
board_build.mcu = stm32f103vct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103VC (48k RAM, 256k Flash) supports the next uploading protocols:
You can change upload protocol using *upload_protocol* option:

```ini
[env:genericSTM32F103VC]
platform = ststm32
board = genericSTM32F103VC
upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

STM32F103VC (48k RAM, 256k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103VD (64k RAM. 384k Flash)

Contents

- STM32F103VD (64k RAM. 384k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103VDT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>384KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103VD ID for board option in "platformio.ini" (Project Configuration File):
You can override default STM32F103VD (64k RAM. 384k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103VD.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103VD]
platform = ststm32
board = genericSTM32F103VD

; change microcontroller
board_build.mcu = stm32f103vdt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

## Uploading

STM32F103VD (64k RAM. 384k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103VD]
platform = ststm32
board = genericSTM32F103VD

upload_protocol = stlink
```

## Debugging

`Debugging` - "1-click" solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).
STM32F103VD (64k RAM, 384k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F103VE (64k RAM, 512k Flash)**

**Hardware**

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F103VE` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F103VE]
platform = ststm32
board = genericSTM32F103VE
```

You can override default STM32F103VE (64k RAM, 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103VE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103VE]
platform = ststm32
board = genericSTM32F103VE

; change microcontroller
board_build.mcu = stm32f103vet6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103VE (64k RAM, 512k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103VE]
platform = ststm32
board = genericSTM32F103VE

upload_protocol = stlink
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

STM32F103VE (64k RAM, 512k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F103VF (96k RAM, 768k Flash)**

**Contents**

- **STM32F103VF (96k RAM, 768k Flash)**
  - Hardware
  - Configuration
  - Uploading
PlatformIO Documentation, Release 5.0.5a1

- Debugging
- Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103VF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>768KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103VF` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F103VF]
platform = ststm32
board = genericSTM32F103VF
```

You can override default STM32F103VF (96k RAM. 768k Flash) settings per build environment using `board_{***}` option, where `{***}` is a JSON object path from board manifest `genericSTM32F103VF.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103VF]
platform = ststm32
board = genericSTM32F103VF

; change microcontroller
board_build.mcu = stm32f103vf

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103VF (96k RAM. 768k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink
Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:genericSTM32F103VF]
platform = ststm32
board = genericSTM32F103VF
upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103VF (96k RAM, 768k Flash) does not have on-board debug probe and **IS NOT READY** for debugging.

You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F103VG (96k RAM, 1024k Flash)**
Contents

- STM32F103VG (96k RAM. 1024k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103VG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103VG` ID for `board` option in `"platformio.ini" (Project Configuration File):

```
[env:genericSTM32F103VG]
platform = ststm32
board = genericSTM32F103VG
```

You can override default STM32F103VG (96k RAM. 1024k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103VG.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```
[env:genericSTM32F103VG]
platform = ststm32
board = genericSTM32F103VG

; change microcontroller
board_build.mcu = stm32f103vg

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103VG (96k RAM. 1024k Flash) supports the next uploading protocols:
• blackmagic
• dfu
• jlink
• serial
• stlink

Default protocol is serial

You can change upload protocol using upload_protocol option:

```ini
[env:genericSTM32F103VG]
platform = ststm32
board = genericSTM32F103VG
upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F103VG (96k RAM, 1024k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tī Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103ZC (48k RAM. 256k Flash)

Contents

- STM32F103ZC (48k RAM. 256k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103ZCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>48KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103ZC ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F103ZC (48k RAM. 256k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103ZC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103ZC]
platform = ststm32
board = genericSTM32F103ZC

; change microcontroller
board_build.mcu = stm32f103zct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32F103ZC (48k RAM. 256k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103ZC]
platform = ststm32
board = genericSTM32F103ZC

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).
STM32F103ZC (48k RAM, 256k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103ZD (64k RAM, 384k Flash)

Contents

- STM32F103ZD (64k RAM, 384k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103ZDT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>384KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F103ZD` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```python
[env:genericSTM32F103ZD]
platform = ststm32
board = genericSTM32F103ZD
```

You can override default STM32F103ZD (64k RAM, 384k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103ZD.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:genericSTM32F103ZD]
platform = ststm32
board = genericSTM32F103ZD

; change microcontroller
board_build.mcu = stm32f103zdt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103ZD (64k RAM, 384k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```python
[env:genericSTM32F103ZD]
platform = ststm32
board = genericSTM32F103ZD

upload_protocol = stlink
```
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103ZD (64k RAM, 384k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F103ZE (64k RAM. 512k Flash)

Contents

- **STM32F103ZE (64k RAM. 512k Flash)**
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103ZET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F103ZE ID for board option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F103ZE]
platform = ststm32
board = genericSTM32F103ZE
```

You can override default STM32F103ZE (64k RAM, 512k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F103ZE.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:genericSTM32F103ZE]
platform = ststm32
board = genericSTM32F103ZE

; change microcontroller
board_build.mcu = stm32f103zet6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103ZE (64k RAM, 512k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink
Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103ZE]
platform = ststm32
board = genericSTM32F103ZE
upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103ZE (64k RAM, 512k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F103ZF (96k RAM, 768k Flash)

Contents

- STM32F103ZF (96k RAM, 768k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103ZF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>768KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F103ZF` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:genericSTM32F103ZF]
platform = ststm32
board = genericSTM32F103ZF
```

You can override default STM32F103ZF (96k RAM, 768k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F103ZF.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F103ZF]
platform = ststm32
board = genericSTM32F103ZF

; change microcontroller
board_build.mcu = stm32f103zf

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

STM32F103ZF (96k RAM. 768k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103ZF]
platform = ststm32
board = genericSTM32F103ZF
upload_protocol = serial
```

Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F103ZF (96k RAM. 768k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

**STM32F103ZG (96k RAM. 1024k Flash)**

**Contents**

- STM32F103ZG (96k RAM. 1024k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103ZG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F103ZG` ID for `board` option in "platformio.ini" (Project Configuration File):
You can override default STM32F103ZG (96k RAM. 1024k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103ZG.json`. For example, `board_build.mcu, board_build.f_cpu, etc.

```ini
[env:genericSTM32F103ZG]
platform = ststm32
board = genericSTM32F103ZG

; change microcontroller
board_build.mcu = stm32f103zg

; change MCU frequency
board_build.f_cpu = 72000000L
```

**Uploading**

STM32F103ZG (96k RAM. 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103ZG]
platform = ststm32
board = genericSTM32F103ZG

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

*Warning:* You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File)*.

STM32F103ZG (96k RAM. 1024k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | |
J-LINK | | |
ST-LINK | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F303CB (32k RAM. 128k Flash)

Contents

- STM32F303CB (32k RAM. 128k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `genericSTM32F303CB` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F303CB]
platform = ststm32
board = genericSTM32F303CB
```

You can override default STM32F303CB (32k RAM, 128k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F303CB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F303CB]
platform = ststm32
board = genericSTM32F303CB

; change microcontroller
board_build.mcu = stm32f303cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32F303CB (32k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F303CB]
platform = ststm32
board = genericSTM32F303CB

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F303CB (32k RAM, 128k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F401CB (64k RAM, 128k Flash)

Contents

- STM32F401CB (64k RAM, 128k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F401CB` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:genericSTM32F401CB]
platform = ststm32
board = genericSTM32F401CB
```

You can override default STM32F401CB (64k RAM, 128k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F401CB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F401CB]
platform = ststm32
board = genericSTM32F401CB

; change microcontroller
board_build.mcu = stm32f401cb

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

STM32F401CB (64k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
[env:genericSTM32F401CB]
platform = ststm32
board = genericSTM32F401CB
upload_protocol = serial

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

STM32F401CB (64k RAM, 128k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F401CC (64k RAM, 256k Flash)**
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F401CC` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F401CC]
platform = ststm32
board = genericSTM32F401CC
```

You can override default STM32F401CC (64k RAM. 256k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F401CC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F401CC]
platform = ststm32
board = genericSTM32F401CC

; change microcontroller
board_build.mcu = stm32f401cc

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

STM32F401CC (64k RAM. 256k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F401CC]
platform = ststm32
board = genericSTM32F401CC
upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

STM32F401CC (64k RAM. 256k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F401CD (96k RAM. 384k Flash)

Contents

- STM32F401CD (96k RAM. 384k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>384KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F401CD` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:genericSTM32F401CD]
platform = ststm32
board = genericSTM32F401CD
```

You can override default STM32F401CD (96k RAM. 384k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F401CD.json`. For example, `board_build.mcu, board_build.f_cpu` etc.

```
[env:genericSTM32F401CD]
platform = ststm32
board = genericSTM32F401CD

; change microcontroller
board_build.mcu = stm32f401cd

; change MCU frequency
board_build.f_cpu = 84000000L
```
Uploading

STM32F401CD (96k RAM, 384k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F401CD]
platform = ststm32
board = genericSTM32F401CD
upload_protocol = serial
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F401CD (96k RAM, 384k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F401CE (96k RAM. 512k Flash)

Contents

- STM32F401CE (96k RAM. 512k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F401CE ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F401CE (96k RAM, 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F401CE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F401CE]
platform = ststm32
board = genericSTM32F401CE

; change microcontroller
board_build.mcu = stm32f401ce

; change MCU frequency
board_build.f_cpu = 84000000L
```

### Uploading

STM32F401CE (96k RAM, 512k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F401CE]
platform = ststm32
board = genericSTM32F401CE

upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F401CE (96k RAM, 512k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | |
J-LINK | | |
ST-LINK | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F401RB (64k RAM. 128k Flash)

Contents

- STM32F401RB (64k RAM. 128k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `genericSTM32F401RB` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F401RB]
platform = ststm32
board = genericSTM32F401RB
```

You can override default STM32F401RB (64k RAM, 128k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F401RB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F401RB]
platform = ststm32
board = genericSTM32F401RB

; change microcontroller
board_build.mcu = stm32f401rb

; change MCU frequency
board_build.f_cpu = 84000000L
```

### Uploading

STM32F401RB (64k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F401RB]
platform = ststm32
board = genericSTM32F401RB

upload_protocol = serial
```

### Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F401RB (64k RAM. 128k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F401RC (64k RAM. 256k Flash)

Contents

- STM32F401RC (64k RAM. 256k Flash)
  - Hardware
  - Configuration
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F401RC` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:genericSTM32F401RC]
platform = ststm32
board = genericSTM32F401RC
```

You can override default STM32F401RC (64k RAM, 256k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F401RC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F401RC]
platform = ststm32
board = genericSTM32F401RC

; change microcontroller
board_build.mcu = stm32f401rc

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

STM32F401RC (64k RAM, 256k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink
Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F401RC]
platform = ststm32
board = genericSTM32F401RC
upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F401RC (64k RAM, 256k Flash) does not have on-board debug probe and is not ready for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### STM32F401RD (96k RAM. 384k Flash)

#### Contents

- **STM32F401RD (96k RAM. 384k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `genericSTM32F401RD` ID for `board` option in "`platformio.ini` (Project Configuration File):

```ini
[env:genericSTM32F401RD]
platform = ststm32
board = genericSTM32F401RD
```

You can override default STM32F401RD (96k RAM, 384k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F401RD.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F401RD]
platform = ststm32
board = genericSTM32F401RD

; change microcontroller
board_build.mcu = stm32f401rd

; change MCU frequency
board_build.f_cpu = 84000000L
```

### Uploading

STM32F401RD (96k RAM, 384k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F401RD]
platform = ststm32
board = genericSTM32F401RD

upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F401RD (96k RAM, 384k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F401RE (96k RAM, 512k Flash)
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>84MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>96kB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F401RE` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:genericSTM32F401RE]
platform = ststm32
board = genericSTM32F401RE
```

You can override default STM32F401RE (96k RAM, 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F401RE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F401RE]
platform = ststm32
board = genericSTM32F401RE

; change microcontroller
board_build.mcu = stm32f401re

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

STM32F401RE (96k RAM, 512k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
[env:genericSTM32F401RE]
platform = ststm32
board = genericSTM32F401RE
upload_protocol = serial

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F401RE (96k RAM, 512k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F405RG (128k RAM. 1024k Flash)

Contents

- STM32F405RG (128k RAM. 1024k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F405RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F405RG` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F405RG]
platform = ststm32
board = genericSTM32F405RG
```

You can override default STM32F405RG (128k RAM. 1024k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F405RG.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F405RG]
platform = ststm32
board = genericSTM32F405RG

; change microcontroller
board_build.mcu = stm32f405rg

; change MCU frequency
board_build.f_cpu = 168000000L
```
Uploading

STM32F405RG (128k RAM. 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is serial

You can change upload protocol using upload_protocol option:

```
[env:genericSTM32F405RG]
platform = ststm32
board = genericSTM32F405RG
upload_protocol = serial
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F405RG (128k RAM. 1024k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F407VE (192k RAM, 512k Flash)

Contents

- STM32F407VE (192k RAM, 512k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>502.23KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F407VET6` ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F407VE (192k RAM, 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F407VET6.json`. For example, `board_build.mcu, board_build.f_cpu, etc.`

```ini
[env:genericSTM32F407VET6]
platform = ststm32
board = genericSTM32F407VET6

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

STM32F407VE (192k RAM, 512k Flash) supports the next uploading protocols:

- dfu
- jlink
- stlink

Default protocol is stlink.

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F407VET6]
platform = ststm32
board = genericSTM32F407VET6

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).

STM32F407VE (192k RAM, 512k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

1.12. Boards
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td><strong>CMSIS</strong></td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td><strong>libopencm3</strong></td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

STM32F407VG (192k RAM. 1024k Flash)

Contents

- STM32F407VG (192k RAM. 1024k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F407VGT6` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default STM32F407VG (192k RAM. 1024k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F407VGT6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F407VGT6]
platform = ststm32
board = genericSTM32F407VGT6

; change microcontroller
board_build.mcu = stm32f407vgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

STM32F407VG (192k RAM. 1024k Flash) supports the next uploading protocols:

- dfu
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F407VGT6]
platform = ststm32
board = genericSTM32F407VGT6

upload_protocol = stlink
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F407VG (192k RAM. 1024k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

1.12. Boards

2261
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F410C8 (32k RAM. 64k Flash)

Contents

- STM32F410C8 (32k RAM. 64k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F410C8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use **genericSTM32F410C8** ID for **board** option in "platformio.ini" (Project Configuration File):
You can override default STM32F410C8 (32k RAM, 64k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F410C8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F410C8]
platform = ststm32
board = genericSTM32F410C8

; change microcontroller
board_build.mcu = stm32f410c8

; change MCU frequency
board_build.f_cpu = 100000000L
```

**Uploading**

STM32F410C8 (32k RAM, 64k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F410C8]
platform = ststm32
board = genericSTM32F410C8

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*). STM32F410C8 (32k RAM, 64k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
## Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM32F410CB (32k RAM, 128k Flash)

#### Contents

- **STM32F410CB (32k RAM, 128k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F410CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F410CB` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F410CB]
platform = ststm32
board = genericSTM32F410CB
```

You can override default STM32F410CB (32k RAM. 128k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F410CB.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:genericSTM32F410CB]
platform = ststm32
board = genericSTM32F410CB

; change microcontroller
board_build.mcu = stm32f410cb

; change MCU frequency
board_build.f_cpu = 100000000L
```

**Uploading**

STM32F410CB (32k RAM. 128k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F410CB]
platform = ststm32
board = genericSTM32F410CB

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F410CB (32k RAM. 128k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F410R8 (32k RAM. 64k Flash)
Hardware

Platform *STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F410R8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F410R8 ID for *board* option in “platformio.ini” (Project Configuration File):

```ini
[env:genericSTM32F410R8]
platform = ststm32
board = genericSTM32F410R8
```

You can override default STM32F410R8 (32k RAM, 64k Flash) settings per build environment using *board_*** option, where *** is a JSON object path from board manifest genericSTM32F410R8.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:genericSTM32F410R8]
platform = ststm32
board = genericSTM32F410R8

; change microcontroller
board_build.mcu = stm32f410r8

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

STM32F410R8 (32k RAM, 64k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is *serial*

You can change upload protocol using *upload_protocol* option:
**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

STM32F410R8 (32k RAM. 64k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F410RB (32k RAM. 128k Flash)**

**Contents**
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F410RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F410RB` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:genericSTM32F410RB]
platform = ststm32
board = genericSTM32F410RB
```

You can override default STM32F410RB (32k RAM, 128k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F410RB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F410RB]
platform = ststm32
board = genericSTM32F410RB

; change microcontroller
board_build.mcu = stm32f410rb

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

STM32F410RB (32k RAM, 128k Flash) supports the next uploading protocols:

- blackmagic
- dfu
PlatformIO Documentation, Release 5.0.5a1

- jlink
- serial
- stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F410RB]
platform = ststm32
board = genericSTM32F410RB
upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F410RB (32k RAM. 128k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F411CC (128k RAM. 256k Flash)

Contents

- STM32F411CC (128k RAM. 256k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F411CC ID for board option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F411CC]
platform = ststm32
board = genericSTM32F411CC
```

You can override default STM32F411CC (128k RAM. 256k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F411CC.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:genericSTM32F411CC]
platform = ststm32
board = genericSTM32F411CC

; change microcontroller
board_build.mcu = stm32f411cc

; change MCU frequency
board_build.f_cpu = 100000000L
```
Uploading

STM32F411CC (128k RAM, 256k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is **serial**

You can change upload protocol using *upload_protocol* option:

```ini
[env:genericSTM32F411CC]
platform = ststm32
board = genericSTM32F411CC
upload_protocol = serial
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (*Project Configuration File)*.

STM32F411CC (128k RAM, 256k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F411CE (128k RAM. 512k Flash)

Contents

- STM32F411CE (128k RAM. 512k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F411CE ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F411CE (128k RAM, 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F411CE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F411CE]
platform = ststm32
board = genericSTM32F411CE

; change microcontroller
board_build.mcu = stm32f411ce

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

STM32F411CE (128k RAM, 512k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F411CE]
platform = ststm32
board = genericSTM32F411CE

upload_protocol = serial
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F411CE (128k RAM, 512k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.
## Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM32F411RC (128k RAM. 256k Flash)

#### Contents

- STM32F411RC (128k RAM. 256k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `genericSTM32F411RC` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F411RC]
platform = ststm32
board = genericSTM32F411RC
```

You can override default STM32F411RC (128k RAM, 256k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F411RC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F411RC]
platform = ststm32
board = genericSTM32F411RC

; change microcontroller
board_build.mcu = stm32f411rc

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

STM32F411RC (128k RAM, 256k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F411RC]
platform = ststm32
board = genericSTM32F411RC

upload_protocol = serial
```

### Debugging

Debugging - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F411RC (128k RAM. 256k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CM-SIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F411RE (128k RAM. 256k Flash)

Contents

- STM32F411RE (128k RAM. 256k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F411RE ID for board option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F411RE]
platform = ststm32
board = genericSTM32F411RE
```

You can override default STM32F411RE (128k RAM, 256k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F411RE.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:genericSTM32F411RE]
platform = ststm32
board = genericSTM32F411RE

; change microcontroller
board_build.mcu = stm32f411re

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

STM32F411RE (128k RAM, 256k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is serial

You can change upload protocol using upload_protocol option:
[env:genericSTM32F411RE]
platform = ststm32
board = genericSTM32F411RE
upload_protocol = serial

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

STM32F411RE (128k RAM, 256k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>CMSIS</strong></td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td><strong>libopenCM3</strong></td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F412CE (256k RAM, 512k Flash)**

**Contents**

1.12. Boards 2279
Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F412CE` ID for `board` option in ”`platformio.ini` (Project Configuration File):

```ini
[env:genericSTM32F412CE]
platform = ststm32
board = genericSTM32F412CE
```

You can override default STM32F412CE (256k RAM. 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F412CE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F412CE]
platform = ststm32
board = genericSTM32F412CE

; change microcontroller
board_build.mcu = stm32f412ce

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

STM32F412CE (256k RAM. 512k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is **serial**

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F412CE]
platform = ststm32
board = genericSTM32F412CE
upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F412CE (256k RAM, 512k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F412CG ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:genericSTM32F412CG]
platform = ststm32
board = genericSTM32F412CG
```

You can override default STM32F412CG (256k RAM. 1024k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F412CG.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:genericSTM32F412CG]
platform = ststm32
board = genericSTM32F412CG

; change microcontroller
board_build.mcu = stm32f412cg

; change MCU frequency
board_build.f_cpu = 100000000L
```
Uploading

STM32F412CG (256k RAM, 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is serial.

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F412CG]
platform = ststm32
board = genericSTM32F412CG
upload_protocol = serial
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F412CG (256k RAM, 1024k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F412RE (256k RAM. 512k Flash)

Contents

- STM32F412RE (256k RAM. 512k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F412RE ID for board option in “platformio.ini” (Project Configuration File):
You can override default STM32F412RE (256k RAM. 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F412RE.json`. For example, board_build.mcu, board_build.f_cpu, etc.

```python
[env:genericSTM32F412RE]
platform = ststm32
board = genericSTM32F412RE

; change microcontroller
board_build.mcu = stm32f412re

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

STM32F412RE (256k RAM. 512k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```python
[env:genericSTM32F412RE]
platform = ststm32
board = genericSTM32F412RE

upload_protocol = serial
```

### Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F412RE (256k RAM. 512k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | |
J-LINK | | |
ST-LINK | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F412RG (256k RAM. 1024k Flash)

Contents

- STM32F412RG (256k RAM. 1024k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F412RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F412RG` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F412RG]
platform = ststm32
board = genericSTM32F412RG
```

You can override default STM32F412RG (256k RAM, 1024k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F412RG.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F412RG]
platform = ststm32
board = genericSTM32F412RG

; change microcontroller
board_build.mcu = stm32f412rg

; change MCU frequency
board_build.f_cpu = 100000000L
```

**Uploading**

STM32F412RG (256k RAM, 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F412RG]
platform = ststm32
board = genericSTM32F412RG

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F412RG (256k RAM. 1024k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F413CG (320k RAM. 1024k Flash)

Contents

- STM32F413CG (320k RAM. 1024k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F413CG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F413CG` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:genericSTM32F413CG]
platform = ststm32
board = genericSTM32F413CG
```

You can override default STM32F413CG (320k RAM, 1024k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F413CG.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F413CG]
platform = ststm32
board = genericSTM32F413CG

; change microcontroller
board_build.mcu = stm32f413cg

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

STM32F413CG (320k RAM, 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F413CG (320k RAM, 1024k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STMicroelectronics’ STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F413CH (320k RAM, 1536k Flash)
- **STM32F413CH (320k RAM, 1536k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F413CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.50MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

### Configuration

Please use `genericSTM32F413CH` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F413CH]
platform = ststm32
board = genericSTM32F413CH
```

You can override default STM32F413CH (320k RAM, 1536k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F413CH.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F413CH]
platform = ststm32
board = genericSTM32F413CH

; change microcontroller
board_build.mcu = stm32f413ch

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

STM32F413CH (320k RAM, 1536k Flash) supports the next uploading protocols:

- blackmagic
- dfu
• jlink
• serial
• stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F413CH]
platform = ststm32
board = genericSTM32F413CH
upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F413CH (320k RAM. 1536k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F413RG (320k RAM. 1024k Flash)

Contents

- STM32F413RG (320k RAM. 1024k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F413RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F413RG ID for board option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F413RG]
platform = ststm32
board = genericSTM32F413RG
```

You can override default STM32F413RG (320k RAM. 1024k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F413RG.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:genericSTM32F413RG]
platform = ststm32
board = genericSTM32F413RG

; change microcontroller
board_build.mcu = stm32f413rg

; change MCU frequency
board_build.f_cpu = 100000000L
```
Uploading

STM32F413RG (320k RAM. 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F413RG]
platform = ststm32
board = genericSTM32F413RG
upload_protocol = serial
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F413RG (320k RAM. 1024k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F413RH (320k RAM. 1536k Flash)

Contents

- STM32F413RH (320k RAM. 1536k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F413RH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.50MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F413RH ID` for `board` option in “platformio.ini” (Project Configuration File):
You can override default STM32F413RH (320k RAM. 1536k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F413RH.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```plaintext
[env:genericSTM32F413RH]
platform = ststm32
board = genericSTM32F413RH

; change microcontroller
board_build.mcu = stm32f413rh

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

STM32F413RH (320k RAM. 1536k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:genericSTM32F413RH]
platform = ststm32
board = genericSTM32F413RH

upload_protocol = serial
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F413RH (320k RAM. 1536k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe | Yes | |
J-LINK | | |
ST-LINK | | |

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM32F415RG (128k RAM, 1024k Flash)

#### Contents

- **STM32F415RG (128k RAM, 1024k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F415RG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F415RG` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:genericSTM32F415RG]
platform = ststm32
board = genericSTM32F415RG
```

You can override default STM32F415RG (128k RAM, 1024k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F415RG.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F415RG]
platform = ststm32
board = genericSTM32F415RG

; change microcontroller
board_build.mcu = stm32f415rg

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

STM32F415RG (128k RAM, 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F415RG]
platform = ststm32
board = genericSTM32F415RG

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F415RG (128k RAM. 1024k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F417VE (128k RAM. 512k Flash)**

**Contents**

- **STM32F417VE (128k RAM. 512k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F417VE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F417VE` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:genericSTM32F417VE]
platform = ststm32
board = genericSTM32F417VE
```

You can override default STM32F417VE (128k RAM. 512k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F417VE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F417VE]
platform = ststm32
board = genericSTM32F417VE

; change microcontroller
board_build.mcu = stm32f417ve

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

STM32F417VE (128k RAM. 512k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F417VE (128k RAM. 512k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F417VG (128k RAM. 1024k Flash)
PlatformIO Documentation, Release 5.0.5a1

- **STM32F417VG (128k RAM, 1024k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F417VG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `genericSTM32F417VG` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:genericSTM32F417VG]
platform = ststm32
board = genericSTM32F417VG
```

You can override default STM32F417VG (128k RAM, 1024k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F417VG.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F417VG]
platform = ststm32
board = genericSTM32F417VG

; change microcontroller
board_build.mcu = stm32f417vg

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

STM32F417VG (128k RAM, 1024k Flash) supports the next uploading protocols:

- blackmagic
- dfu
• jlink
• serial
• stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F417VG]
platform = ststm32
board = genericSTM32F417VG

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “`platformio.ini` (Project Configuration File).

STM32F417VG (128k RAM. 1024k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32F423CH (320k RAM. 1536k Flash)

Contents

• STM32F423CH (320k RAM. 1536k Flash)
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F423CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.50MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use genericSTM32F423CH ID for board option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F423CH]
platform = ststm32
board = genericSTM32F423CH
```

You can override default STM32F423CH (320k RAM. 1536k Flash) settings per build environment using board_*** option, where *** is a JSON object path from board manifest genericSTM32F423CH.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:genericSTM32F423CH]
platform = ststm32
board = genericSTM32F423CH

; change microcontroller
board_build.mcu = stm32f423ch

; change MCU frequency
board_build.f_cpu = 100000000L
```
Uploading

STM32F423CH (320k RAM, 1536k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F423CH]
platform = ststm32
board = genericSTM32F423CH
upload_protocol = serial
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F423CH (320k RAM, 1536k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F423RH (320k RAM. 1536k Flash)

Contents

- **STM32F423RH (320k RAM. 1536k Flash)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F423RH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.50MB</td>
</tr>
<tr>
<td>RAM</td>
<td>320KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F423RH` ID for `board` option in “platformio.ini” (Project Configuration File):
You can override default STM32F423RH (320k RAM. 1536k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F423RH.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F423RH]
platform = ststm32
board = genericSTM32F423RH

; change microcontroller
board_build.mcu = stm32f423rh

; change MCU frequency
board_build.f_cpu = 100000000L
```

**Uploading**

STM32F423RH (320k RAM. 1536k Flash) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F423RH]
platform = ststm32
board = genericSTM32F423RH

upload_protocol = serial
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F423RH (320k RAM. 1536k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
### Compatible Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, TI Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### STM32F446RC (128k RAM. 256k Flash)

**Contents**

- STM32F446RC (128k RAM. 256k Flash)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform **STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `genericSTM32F446RC` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:genericSTM32F446RC]
platform = ststm32
board = genericSTM32F446RC
```

You can override default STM32F446RC (128k RAM, 256k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F446RC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:genericSTM32F446RC]
platform = ststm32
board = genericSTM32F446RC

; change microcontroller
board_build.mcu = stm32f446rc

; change MCU frequency
board_build.f_cpu = 180000000L
```

### Uploading

STM32F446RC (128k RAM, 256k Flash) supports the next uploading protocols:
- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F446RC]
platform = ststm32
board = genericSTM32F446RC

upload_protocol = serial
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

STM32F446RC (128k RAM. 256k Flash) does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F446RE (128k RAM. 512k Flash)**

**Contents**

- *STM32F446RE (128k RAM. 512k Flash)*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F446RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F446RE` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:genericSTM32F446RE]
platform = ststm32
board = genericSTM32F446RE
```

You can override default STM32F446RE (128k RAM. 512k Flash) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `genericSTM32F446RE.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:genericSTM32F446RE]
platform = ststm32
board = genericSTM32F446RE

; change microcontroller
board_build.mcu = stm32f446re

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

STM32F446RE (128k RAM. 512k Flash) supports the next uploading protocols:

- `blackmagic`
- `dfu`
- `jlink`
- `serial`
- `stlink`

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:
Debugging

“1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

STM32F446RE (128k RAM, 512k Flash) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>liopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

STM32F4Stamp F405
STM32F4Stamp F405

- **Hardware**
- **Configuration**
- **Uploading**
- **Debugging**
- **Frameworks**

**Hardware**

Platform *ST STM32*: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F405RGT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

**Configuration**

Please use *stm32f4stamp* ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:stm32f4stamp]
platform = ststm32
board = stm32f4stamp
```

You can override default STM32F4Stamp F405 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest *stm32f4stamp.json*. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stm32f4stamp]
platform = ststm32
board = stm32f4stamp

; change microcontroller
board_build.mcu = stm32f405rgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

STM32F4Stamp F405 supports the next uploading protocols:

- dfu
- jlink
• stlink

Default protocol is dfu

You can change upload protocol using `upload_protocol` option:

```
[env:stm32f4stamp]
platform = stm32
board = stm32f4stamp
upload_protocol = dfu
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (Project Configuration File).

STM32F4Stamp F405 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tî Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**STM32F7508-DK**
Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F750N8H6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>216MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>64KB</td>
</tr>
<tr>
<td>RAM</td>
<td>340KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `disco_f750n8` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:disco_f750n8]
platform = ststm32
board = disco_f750n8
```

You can override default STM32F7508-DK settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `disco_f750n8.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:disco_f750n8]
platform = ststm32
board = disco_f750n8

; change microcontroller
board_build.mcu = stm32f750n8h6

; change MCU frequency
board_build.f_cpu = 216000000L
```

**Uploading**

STM32F7508-DK supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_f750n8]
platform = ststm32
board = disco_f750n8
upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” (Project Configuration File).

STM32F7508-DK has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>li-bopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
STM32H747I-DISCO

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32H747XIH6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>400MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use disco_h747x1 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:disco_h747xi]
platform = ststm32
board = disco_h747xi
```

You can override default STM32H747I-DISCO settings per build environment using board_*** option, where *** is a JSON object path from board manifest disco_h747xi.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:disco_h747xi]
platform = ststm32
board = disco_h747xi

; change microcontroller
board_build.mcu = stm32h747xih6

; change MCU frequency
board_build.f_cpu = 400000000L
```
Uploading

STM32H747I-DISCO supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_h747xi]
platform = stm32
board = disco_h747xi
upload_protocol = stlink
```

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

STM32H747I-DISCO has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>ZephyrRTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

Seeed Arch Max

Contents

- Seeed Arch Max
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F407VET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>192KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>
Configuration

Please use `seeedArchMax ID` for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:seeedArchMax]
platform = ststm32
board = seeedArchMax
```

You can override default Seeed Arch Max settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `seeedArchMax.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:seeedArchMax]
platform = ststm32
board = seeedArchMax

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

Seeed Arch Max supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:seeedArchMax]
platform = ststm32
board = seeedArchMax

upload_protocol = mbed
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini` (Project Configuration File).
Seeed Arch Max has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

Seeed Wio 3G

Contents

- Seeed Wio 3G
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F439VI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>SeeedStudio</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `wio_3g` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:wio_3g]
platform = ststm32
board = wio_3g
```

You can override default Seeed Wio 3G settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `wio_3g.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:wio_3g]
platform = ststm32
board = wio_3g

; change microcontroller
board_build.mcu = stm32f439vi

; change MCU frequency
board_build.f_cpu = 180000000L
```

**Uploading**

Seeed Wio 3G supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:wio_3g]
platform = ststm32
board = wio_3g

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Seeed Wio 3G has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3 / M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

SensorTile.box

Contents

- SensorTile.box
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
PlatformIO Documentation, Release 5.0.5a1

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L4R9ZI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>640KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `steval_mksboxv1` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:steval_mksboxv1]
platform = ststm32
board = steval_mksboxv1
```

You can override default SensorTile.box settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `steval_mksboxv1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:steval_mksboxv1]
platform = ststm32
board = steval_mksboxv1

; change microcontroller
board_build.mcu = stm32l4r9zi

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

SensorTile.box supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

SensorTile.box does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### Sparky V1 F303
Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F303CCT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>40KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TauLabs</td>
</tr>
</tbody>
</table>

Configuration

Please use `sparky_v1` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:sparky_v1]
platform = ststm32
board = sparky_v1
```

You can override default Sparky V1 F303 settings per build environment using board_*** option, where *** is a JSON object path from board manifest `sparky_v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparky_v1]
platform = ststm32
board = sparky_v1

; change microcontroller
board_build.mcu = stm32f303cct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Sparky V1 F303 supports the next uploading protocols:
• blackmagic
• cmsis-dap
• jlink
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:sparky_v1]
platform = ststm32
board = sparky_v1

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

***Warning:*** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Sparky V1 F303 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

ThunderPack v1.0

Contents

- ThunderPack v1.0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32L072KZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>192KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ThunderPack</td>
</tr>
</tbody>
</table>

Configuration

Please use `thunder_pack ID` for `board` option in “`platformio.ini` (Project Configuration File):”
You can override default ThunderPack v1.0 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `thunder_pack.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:thunder_pack]
platform = ststm32
board = thunder_pack

; change microcontroller
board_build.mcu = stm32l072kz

; change MCU frequency
board_build.f_cpu = 32000000L
```

## Uploading

ThunderPack v1.0 supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:thunder_pack]
platform = ststm32
board = thunder_pack

upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

ThunderPack v1.0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.
**Compatible Tools**

<table>
<thead>
<tr>
<th>Tool</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**ThunderPack v1.1+**

**Contents**

- ThunderPack v1.1+
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `thunder_pack_f411` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[env:thunder_pack_f411]
platform = ststm32
board = thunder_pack_f411
```

You can override default ThunderPack v1.1+ settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `thunder_pack_f411.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:thunder_pack_f411]
platform = ststm32
board = thunder_pack_f411

; change microcontroller
board_build.mcu = stm32f411ceu6

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

ThunderPack v1.1+ supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:thunder_pack_f411]
platform = ststm32
board = thunder_pack_f411

upload_protocol = stlink
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

ThunderPack v1.1+ does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Tiny STM103T

- Tiny STM103T
  - Hardware
  - Configuration
  - Uploading
Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103TBU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>20KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>HY</td>
</tr>
</tbody>
</table>

Configuration

Please use hy_tinystm103tb ID for board option in “platformio.ini” (Project Configuration File):

```
[env:hy_tinystm103tb]
platform = ststm32
board = hy_tinystm103tb
```

You can override default Tiny STM103T settings per build environment using board_*** option, where *** is a JSON object path from board manifest hy_tinystm103tb.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:hy_tinystm103tb]
platform = ststm32
board = hy_tinystm103tb

; change microcontroller
board_build.mcu = stm32f103tbu6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Tiny STM103T supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink
Default protocol is dfu

You can change upload protocol using `upload_protocol` option:

```ini
[env:hy_tinystm103tb]
platform = ststm32
board = hy_tinystm103tb
upload_protocol = dfu
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

Tiny STM103T does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+) / M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
VAkE v1.0

Contents

• VAkE v1.0
  – Hardware
  – Configuration
  – Uploading
  – Debugging
  – Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F446RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>VAE</td>
</tr>
</tbody>
</table>

Configuration

Please use vake_v1 ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:vake_v1]
platform = ststm32
board = vake_v1
```

You can override default VAkE v1.0 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest vake_v1.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:vake_v1]
platform = ststm32
board = vake_v1

; change microcontroller
board_build.mcu = stm32f446ret6

; change MCU frequency
board_build.f_cpu = 180000000L
```
Uploading

VAKE v1.0 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:vake_v1]
platform = ststm32
board = vake_v1
upload_protocol = stlink
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug_tool option in “platformio.ini” *(Project Configuration File)*.

VAKE v1.0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CM-SIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

VCCGND F103ZET6 Mini

Contents

- VCCGND F103ZET6 Mini
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F103ZET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>VCCGND</td>
</tr>
</tbody>
</table>

Configuration

Please use vccgnd_f103zet6 ID for board option in “platformio.ini” (Project Configuration File):
You can override default VCCGND F103ZET6 Mini settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `vccgnd_f103zet6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:vccgnd_f103zet6]
platform = ststm32
board = vccgnd_f103zet6

; change microcontroller
board_build.mcu = stm32f103zet6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

VCCGND F103ZET6 Mini supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:vccgnd_f103zet6]
platform = ststm32
board = vccgnd_f103zet6

upload_protocol = stlink
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).
VCCGND F103ZET6 Mini does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**Waveshare Open103Z**

**Contents**

- **Waveshare Open103Z**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
Microcontroller | STM32F103ZET6  
---|---  
Frequency | 72MHz  
Flash | 512KB  
RAM | 64KB  
Vendor | Waveshare  

## Configuration

Please use waveshare_open103z ID for `board` option in “platformio.ini” (Project Configuration File):

```python
[env:waveshare_open103z]
platform = ststm32
board = waveshare_open103z
```

You can override default Waveshare Open103Z settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `waveshare_open103z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```python
[env:waveshare_open103z]
platform = ststm32
board = waveshare_open103z

; change microcontroller
board_build.mcu = stm32f103zet6

; change MCU frequency
board_build.f_cpu = 72000000L
```

## Uploading

Waveshare Open103Z supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```python
[env:waveshare_open103z]
platform = ststm32
board = waveshare_open103z

upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using **debug_tool** option in “platformio.ini” (Project Configuration File).

Waveshare Open103Z does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**WeAct BlackPill V2.0 (BlackPill F411CE)**

**Contents**

- **WeAct BlackPill V2.0 (BlackPill F411CE)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
PlatformIO Documentation, Release 5.0.5a1

Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411CEU6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WeAct</td>
</tr>
</tbody>
</table>

Configuration

Please use `blackpill_f411ce` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:blackpill_f411ce]
platform = ststm32
board = blackpill_f411ce
```

You can override default WeAct BlackPill V2.0 (BlackPill F411CE) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `blackpill_f411ce.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:blackpill_f411ce]
platform = ststm32
board = blackpill_f411ce

; change microcontroller
board_build.mcu = stm32f411ceu6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

WeAct BlackPill V2.0 (BlackPill F411CE) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

WeAct BlackPill V2.0 (BlackPill F411CE) does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0+/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Wraith V1 ESC

1.12. Boards
PlatformIO Documentation, Release 5.0.5a1

Contents

- Wraith V1 ESC
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F051K6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>7.75KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Airbot</td>
</tr>
</tbody>
</table>

Configuration

Please use wraith32_v1 ID for board option in “platformio.ini” (Project Configuration File):

```plaintext
[env:wraith32_v1]
platform = ststm32
board = wraith32_v1
```

You can override default Wraith V1 ESC settings per build environment using board_*** option, where *** is a JSON object path from board manifest wraith32_v1.json. For example, board_build.mcu, board_build.f_cpu, etc.

```plaintext
[env:wraith32_v1]
platform = ststm32
board = wraith32_v1

; change microcontroller
board_build.mcu = stm32f051k6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Wraith V1 ESC supports the next uploading protocols:
- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is **stlink**

You can change upload protocol using `upload_protocol` option:

```ini
[env:wraith32_v1]
platform = ststm32
board = wraith32_v1
upload_protocol = stlink
```

## Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

Wraith V1 ESC does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

sakura.io Evaluation Board

Contents

- sakura.io Evaluation Board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F411RET6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>100MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>sakura.io</td>
</tr>
</tbody>
</table>

Configuration

Please use sakuraio_evb_01 ID for board option in “platformio.ini” (Project Configuration File):
You can override default sakura.io Evaluation Board settings per build environment using `board_***` option, where *** is a JSON object path from board manifest sakuraio_evb_01.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sakuraio_evb_01]
platform = ststm32
board = sakuraio_evb_01

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

### Uploading

sakura.io Evaluation Board supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:sakuraio_evb_01]
platform = ststm32
board = sakuraio_evb_01

upload_protocol = mbed
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

sakura.io Evaluation Board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### Compatible Tools

<table>
<thead>
<tr>
<th>Black Magic Probe</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32CUBE</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

### u-blox C030-N211 IoT Starter Kit

#### Contents

- u-blox C030-N211 IoT Starter Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

#### Hardware

Platform **ST STM32**: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `ublox_c030_n211` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ublox_c030_n211]
platform = ststm32
board = ublox_c030_n211
```

You can override default u-blox C030-N211 IoT Starter Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ublox_c030_n211.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ublox_c030_n211]
platform = ststm32
board = ublox_c030_n211

; change microcontroller
board_build.mcu = stm32f437vg

; change MCU frequency
board_build.f_cpu = 180000000L
```

### Uploading

u-blox C030-N211 IoT Starter Kit supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:ublox_c030_n211]
platform = ststm32
board = ublox_c030_n211

upload_protocol = mbed
```

### Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

u-blox C030-N211 IoT Starter Kit does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

u-blox C030-R410M IoT

Contents

- u-blox C030-R410M IoT
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
## Configuration

Please use `ublox_c030_r410m` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:ublox_c030_r410m]
platform = ststm32
board = ublox_c030_r410m
```

You can override default u-blox C030-R410M IoT settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `ublox_c030_r410m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ublox_c030_r410m]
platform = ststm32
board = ublox_c030_r410m

; change microcontroller
board_build.mcu = stm32f437vg

; change MCU frequency
board_build.f_cpu = 180000000L
```

## Uploading

u-blox C030-R410M IoT supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:ublox_c030_r410m]
platform = ststm32
board = ublox_c030_r410m

upload_protocol = stlink
```

## Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

u-blox C030-R410M IoT has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

u-blox C030-U201 IoT Starter Kit

Contents

- u-blox C030-U201 IoT Starter Kit
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### Configuration

Please use `ublox_c030_u201` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:ublox_c030_u201]
platform = ststm32
board = ublox_c030_u201
```

You can override default u-blox C030-U201 IoT Starter Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ublox_c030_u201.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:ublox_c030_u201]
platform = ststm32
board = ublox_c030_u201

; change microcontroller
board_build.mcu = stm32f437vg

; change MCU frequency
board_build.f_cpu = 180000000L
```

### Uploading

u-blox C030-U201 IoT Starter Kit supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:ublox_c030_u201]
platform = ststm32
board = ublox_c030_u201

upload_protocol = mbed
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

u-blox C030-U201 IoT Starter Kit does not have on-board debug probe and is NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

u-blox EVK-ODIN-W2

Contents

- u-blox EVK-ODIN-W2
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
### PlatformIO Documentation, Release 5.0.5a1

**Microcontroller**
<table>
<thead>
<tr>
<th></th>
<th>STM32F439ZIY6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frequency</strong></td>
<td>168MHz</td>
</tr>
<tr>
<td><strong>Flash</strong></td>
<td>2MB</td>
</tr>
<tr>
<td><strong>RAM</strong></td>
<td>256KB</td>
</tr>
<tr>
<td><strong>Vendor</strong></td>
<td>u-blox</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `ublox_evk_odin_w2` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:ublox_evk_odin_w2]
platform = ststm32
board = ublox_evk_odin_w2
```

You can override default u-blox EVK-ODIN-W2 settings per build environment using `board_{***}` option, where `***` is a JSON object path from board manifest `ublox_evk_odin_w2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ublox_evk_odin_w2]
platform = ststm32
board = ublox_evk_odin_w2

; change microcontroller
board_build.mcu = stm32f439ziy6

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

u-blox EVK-ODIN-W2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:ublox_evk_odin_w2]
platform = ststm32
board = ublox_evk_odin_w2

upload_protocol = stlink
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

u-blox EVK-ODIN-W2 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

u-blox ODIN-W2

Contents

- u-blox ODIN-W2
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM32: The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F439ZIY6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>u-blox</td>
</tr>
</tbody>
</table>

### Configuration

Please use `mtb_ublox_odin_w2` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:mtb_ublox_odin_w2]
platform = ststm32
board = mtb_ublox_odin_w2
```

You can override default u-blox ODIN-W2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mtb_ublox_odin_w2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mtb_ublox_odin_w2]
platform = ststm32
board = mtb_ublox_odin_w2

; change microcontroller
board_build.mcu = stm32f439ziy6

; change MCU frequency
board_build.f_cpu = 168000000L
```

### Uploading

u-blox ODIN-W2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:mtb_ublox_odin_w2]
platform = ststm32
board = mtb_ublox_odin_w2

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

u-blox ODIN-W2 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black Magic Probe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
</tbody>
</table>

1.12.29 ST STM8

ST STM8S-DISCOVERY

Contents

- ST STM8S-DISCOVERY
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM8: The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.
PlatformIO Documentation, Release 5.0.5a1

### Configuration

Please use `stm8sdisco` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:stm8sdisco]
platform = ststm8
board = stm8sdisco
```

You can override default ST STM8S-DISCOVERY settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `stm8sdisco.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:stm8sdisco]
platform = ststm8
board = stm8sdisco

; change microcontroller
board_build.mcu = stm8s105c6t6

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Uploading

ST STM8S-DISCOVERY supports the next uploading protocols:

- serial
- stlink
- stlinkv2

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:stm8sdisco]
platform = ststm8
board = stm8sdisco

upload_protocol = stlink
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM8S105C6T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

ST STM8S-DISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST-LINK</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>Standard Peripheral Library</strong></td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
</tbody>
</table>

**ST STM8S103F3 Breakout Board**

**Contents**

- **ST STM8S103F3 Breakout Board**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STM8**: The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM8S103F3P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>
Configuration

Please use \texttt{stm8sblue} ID for \textit{board} option in “\texttt{platformio.ini}” (Project Configuration File):

\begin{verbatim}
[env:stm8sblue]
platform = ststm8
board = stm8sblue
\end{verbatim}

You can override default ST STM8S103F3 Breakout Board settings per build environment using \texttt{board_\*\*\*} option, where \*\*\* is a JSON object path from board manifest \texttt{stm8sblue.json}. For example, \texttt{board_build.mcu}, \texttt{board_build.f_cpu}, etc.

\begin{verbatim}
[env:stm8sblue]
platform = ststm8
board = stm8sblue

; change microcontroller
board_build.mcu = stm8s103f3p6

; change MCU frequency
board_build.f_cpu = 16000000L
\end{verbatim}

Uploading

ST STM8S103F3 Breakout Board supports the next uploading protocols:

- serial
- stlinkv2

Default protocol is \texttt{serial}

You can change upload protocol using \textit{upload_protocol} option:

\begin{verbatim}
[env:stm8sblue]
platform = ststm8
board = stm8sblue

upload_protocol = serial
\end{verbatim}

Debugging

\textit{Debugging} currently does not support ST STM8S103F3 Breakout Board board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
</tbody>
</table>
ST STM8S105K4T6 Breakout Board

Contents

- ST STM8S105K4T6 Breakout Board
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform ST STM8: The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM8S105K4T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use stm8sblack ID for board option in “platformio.ini” (Project Configuration File):

```
[env:stm8sblack]
platform = ststm8
board = stm8sblack
```

You can override default ST STM8S105K4T6 Breakout Board settings per build environment using board_*** option, where *** is a JSON object path from board manifest stm8sblack.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:stm8sblack]
platform = ststm8
board = stm8sblack

; change microcontroller
board_build.mcu = stm8s105k4t6

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

ST STM8S105K4T6 Breakout Board supports the next uploading protocols:
• serial
• stlinkv2

Default protocol is serial

You can change upload protocol using `upload_protocol` option:

```
[env:stm8sblack]
platform = ststm8
board = stm8sblack
upload_protocol = serial
```

### Debugging

*Debugging* currently does not support ST STM8S105K4T6 Breakout Board board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
</tbody>
</table>

### sduino MB (STM8S208MBT6B)

**Contents**

- sduino MB (STM8S208MBT6B)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform *ST STM8*: The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.
<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM8S208MBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>6KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>sduino</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `mb208` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:mb208]
platform = stm8
board = mb208
```

You can override default sduino MB (STM8S208MBT6B) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mb208.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:mb208]
platform = stm8
board = mb208

; change microcontroller
board_build.mcu = stm8s208mbt6

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Uploading**

sduino MB (STM8S208MBT6B) supports the next uploading protocols:

- `serial`
- `stlinkv2`

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```ini
[env:mb208]
platform = stm8
board = mb208

upload_protocol = serial
```

**Debugging**

Debugging currently does not support sduino MB (STM8S208MBT6B) board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
</tbody>
</table>

sduino UNO (STM8S105K6)

Contents

- sduino UNO (STM8S105K6)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **STM8**: The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM8S105K6T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>sduino</td>
</tr>
</tbody>
</table>

Configuration

Please use `s8uno` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:s8uno]
platform = ststm8
board = s8uno
```

You can override default sduino UNO (STM8S105K6) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `s8uno.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.
Uploading

sduino UNO (STM8S105K6) supports the next uploading protocols:

- serial
- stlinkv2

Default protocol is serial

You can change upload protocol using upload_protocol option:

```yaml
[env:s8uno]
platform = ststm8
board = s8uno
upload_protocol = serial
```

Debugging

Debugging currently does not support sduino UNO (STM8S105K6) board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
</tbody>
</table>

1.12.30 Teensy

Teensy 2.0

Contents

- Teensy 2.0
Hardware

Platform Teensy: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA32U4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>31.50KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

Configuration

Please use teensy2 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:teensy2]
platform = teensy
board = teensy2
```

You can override default Teensy 2.0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest teensy2.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:teensy2]
platform = teensy
board = teensy2

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

Teensy 2.0 supports the next uploading protocols:

- teensy-cli
- teensy-gui

Default protocol is teensy-gui

You can change upload protocol using upload_protocol option:
PlatformIO Documentation, Release 5.0.5a1

```
[env:teensy2]
platform = teensy
board = teensy2
upload_protocol = teensy-gui
```

**Debugging**

*Debugging* currently does not support Teensy 2.0 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

---

### Teensy 3.0

**Contents**

- *Teensy 3.0*
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

---

**Hardware**

Platform *Teensy*: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK20DX128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `teensy30` ID for `board` option in “platformio.ini” (*Project Configuration File)*:
You can override default Teensy 3.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `teensy30.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
[env:teensy30]
platform = teensy
board = teensy30

; change microcontroller
board_build.mcu = mk20dx128

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

Teensy 3.0 supports the next uploading protocols:

- `teensy-cli`
- `teensy-gui`

Default protocol is `teensy-gui`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:teensy30]
platform = teensy
board = teensy30

upload_protocol = teensy-gui
```

### Debugging

Debugging currently does not support Teensy 3.0 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Teensy 3.1 / 3.2
Hardware

Platform Teensy: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK20DX256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

Configuration

Please use teensy31 ID for board option in "platformio.ini" (Project Configuration File):

```ini
[env:teensy31]
platform = teensy
board = teensy31
```

You can override default Teensy 3.1 / 3.2 settings per build environment using board_*** option, where *** is a JSON object path from board manifest teensy31.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:teensy31]
platform = teensy
board = teensy31

; change microcontroller
board_build.mcu = mk20dx256

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Teensy 3.1 / 3.2 supports the next uploading protocols:

- jlink
- teensy-cli
PlatformIO Documentation, Release 5.0.5a1

- teensy-gui

Default protocol is teensy-gui

You can change upload protocol using upload_protocol option:

```ini
[env:teensy31]
platform = teensy
board = teensy31
upload_protocol = teensy-gui
```

### Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using debug_tool option in “platformio.ini” (Project Configuration File).

Teensy 3.1 / 3.2 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### Teensy 3.5

**Contents**

- Teensy 3.5
  - Hardware
  - Configuration
  - Uploading

1.12. Boards
Hardware

Platform **Teensy**: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK64FX512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>512KB</td>
</tr>
<tr>
<td>RAM</td>
<td>255.99KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

Configuration

Please use `teensy35` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:teensy35]
platform = teensy
board = teensy35
```

You can override default Teensy 3.5 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `teensy35.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:teensy35]
platform = teensy
board = teensy35

; change microcontroller
board_build.mcu = mk64fx512

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Teensy 3.5 supports the next uploading protocols:

- `jlink`
- `teensy-cli`
- `teensy-gui`

Default protocol is `teensy-gui`

You can change upload protocol using `upload_protocol` option:
Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

Teensy 3.5 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>J-LINK</em></td>
<td></td>
<td><em>Yes</em></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Teensy 3.6

**Contents**

- *Teensy 3.6*
  - *Hardware*
  - *Configuration*
  - *Uploading*
  - *Debugging*
  - *Frameworks*
Hardware

Platform **Teensy**: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MK66FX1M0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>180MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

Configuration

Please use `teensy36` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```
[env:teensy36]
platform = teensy
board = teensy36
```

You can override default Teensy 3.6 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `teensy36.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:teensy36]
platform = teensy
board = teensy36

; change microcontroller
board_build.mcu = mk66fx1m0

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

Teensy 3.6 supports the next uploading protocols:

- jlink
- teensy-cli
- teensy-gui

Default protocol is `teensy-gui`

You can change upload protocol using `upload_protocol` option:

```
[env:teensy36]
platform = teensy
board = teensy36

upload_protocol = teensy-gui
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

Teensy 3.6 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Teensy 4.0

Contents

- Teensy 4.0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Teensy: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.
PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>IMXRT1062</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>600MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.94MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `teensy40` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```ini
[env:teensy40]
platform = teensy
board = teensy40
```

You can override default Teensy 4.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `teensy40.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:teensy40]
platform = teensy
board = teensy40

; change microcontroller
board_build.mcu = imxrt1062

; change MCU frequency
board_build.f_cpu = 600000000L
```

**Uploading**

Teensy 4.0 supports the next uploading protocols:

- jlink
- teensy-gui

Default protocol is `teensy-gui`

You can change upload protocol using `upload_protocol` option:

```ini
[env:teensy40]
platform = teensy
board = teensy40

upload_protocol = teensy-gui
```

**Debugging**

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.
You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

Teensy 4.0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Teensy 4.1

#### Contents

- **Teensy 4.1**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

### Hardware

Platform Teensy: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>IMXRT1062</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>600MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>7.75MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

### Configuration

Please use `teensy41` ID for `board` option in "platformio.ini" (Project Configuration File):
You can override default Teensy 4.1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `teensy41.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:teensy41]
platform = teensy
board = teensy41

; change microcontroller
board_build.mcu = imxrt1062

; change MCU frequency
board_build.f_cpu = 600000000L
```

### Uploading

Teensy 4.1 supports the next uploading protocols:

- jlink
- teensy-gui

Default protocol is `teensy-gui`

You can change upload protocol using `upload_protocol` option:

```ini
[env:teensy41]
platform = teensy
board = teensy41

upload_protocol = teensy-gui
```

### Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “platformio.ini” (Project Configuration File).

Teensy 4.1 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>J-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

Teensy LC

Contents

- Teensy LC
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Teensy: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKL26Z64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>62KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

Configuration

Please use `teensylc` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:teensylc]
platform = teensy
board = teensylc
```

You can override default Teensy LC settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `teensylc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:teensylc]
platform = teensy
board = teensylc
```

(continues on next page)
Uploading

Teensy LC supports the next uploading protocols:

- jlink
- teensy-cli
- teensy-gui

Default protocol is teensy-gui

You can change upload protocol using `upload_protocol` option:

```ini
[env:teensylc]
platform = teensy
board = teensylc
upload_protocol = teensy-gui
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in *“platformio.ini” (Project Configuration File).*

Teensy LC does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>J-LINK</strong></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>
Teensy++ 2.0

Contents

- Teensy++ 2.0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform Teensy: Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT90USB1286</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>127KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Teensy</td>
</tr>
</tbody>
</table>

Configuration

Please use teensy2pp ID for board option in “platformio.ini” (Project Configuration File):

```
[env:teensy2pp]
platform = teensy
board = teensy2pp
```

You can override default Teensy++ 2.0 settings per build environment using board_*** option, where *** is a JSON object path from board manifest teensy2pp.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:teensy2pp]
platform = teensy
board = teensy2pp

; change microcontroller
board_build.mcu = at90usb1286

; change MCU frequency
board_build.f_cpu = 16000000L
```
Uploading

Teensy++ 2.0 supports the next uploading protocols:

- teensy-cli
- teensy-gui

Default protocol is teensy-gui

You can change upload protocol using upload_protocol option:

```yaml
[env:teensy2pp]
platform = teensy
board = teensy2pp
upload_protocol = teensy-gui
```

Debugging

*Debugging* currently does not support Teensy++ 2.0 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

1.12.31 TI MSP430

TI FraunchPad MSP-EXP430FR5739LP

Contents

- TI FraunchPad MSP-EXP430FR5739LP
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *TI MSP430*: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.
Microcontroller | MSP430FR5739  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
<td>Flash</td>
<td>15.37KB</td>
<td>RAM</td>
</tr>
</tbody>
</table>

### Configuration

Please use `lpmsp430fr5739` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:lpmsp430fr5739]
platform = timsp430
board = lpmsp430fr5739
```

You can override default TI FraunChPad MSP-EXP430FR5739LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr5739.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lpmsp430fr5739]
platform = timsp430
board = lpmsp430fr5739

; change microcontroller
board_build.mcu = msp430fr5739

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning**: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in `platformio.ini` (Project Configuration File).

TI FraunChPad MSP-EXP430FR5739LP has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**TI LaunchPad MSP-EXP430F5529LP**

**Contents**

- *TI LaunchPad MSP-EXP430F5529LP*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *TI MSP430*: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430F5529</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>25MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>47KB</td>
</tr>
<tr>
<td>RAM</td>
<td>8KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `lpmsp430f5529` ID for *board* option in "platformio.ini" *(Project Configuration File):*

```
[env:lpmsp430f5529]
platform = timsp430
board = lpmsp430f5529
```

You can override default TI LaunchPad MSP-EXP430F5529LP settings per build environment using *board_*** option, where *** is a JSON object path from board manifest `lpmsp430f5529.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```
[env:lpmsp430f5529]
platform = timsp430
board = lpmsp430f5529
```

; change microcontroller

(continues on next page)
board_build.mcu = msp430f5529
;
change MCU frequency
board_build.f_cpu = 25000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using debug tool option in *“platformio.ini” (Project Configuration File)*.

TI LaunchPad MSP-EXP430F5529LP has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>MSP Debug</em></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**TI LaunchPad MSP-EXP430FR2311LP**

Contents

- *TI LaunchPad MSP-EXP430FR2311LP*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *TI MSP430*: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.
PlatformIO Documentation, Release 5.0.5a1

### Microcontroller

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430FR2311</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>3.75KB</td>
</tr>
<tr>
<td>RAM</td>
<td>1KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

### Configuration

Please use `lpmsp430fr2311` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:lpmsp430fr2311]
platform = timsp430
board = lpmsp430fr2311
```

You can override default TI LaunchPad MSP-EXP430FR2311LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr2311.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:lpmsp430fr2311]
platform = timsp430
board = lpmsp430fr2311

; change microcontroller
board_build.mcu = msp430fr2311

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

TI LaunchPad MSP-EXP430FR2311LP has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSP Debug</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**TI LaunchPad MSP-EXP430FR2433LP**

**Contents**

- *TI LaunchPad MSP-EXP430FR2433LP*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *TI MSP430*: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430FR2433</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>15KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `lpmsp430fr2433` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:lpmsp430fr2433]
platform = timsp430
board = lpmsp430fr2433
```

You can override default TI LaunchPad MSP-EXP430FR2433LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr2433.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```
[env:lpmsp430fr2433]
platform = timsp430
board = lpmsp430fr2433

; change microcontroller
```

(continues on next page)
board_build.mcu = msp430fr2433
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Warning:* You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

TI LaunchPad MSP-EXP430FR2433LP has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**TI LaunchPad MSP-EXP430FR4133LP**

Contents

- **TI LaunchPad MSP-EXP430FR4133LP**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **TI MSP430**: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.
Configuration

Please use `lpmsp430fr4133` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:lpmsp430fr4133]
platform = timsp430
board = lpmsp430fr4133
```

You can override default TI LaunchPad MSP-EXP430FR4133LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr4133.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:lpmsp430fr4133]
platform = timsp430
board = lpmsp430fr4133

; change microcontroller
board_build.mcu = msp430fr4133

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

TI LaunchPad MSP-EXP430FR4133LP has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

TI LaunchPad MSP-EXP430FR5969LP

Contents

- TI LaunchPad MSP-EXP430FR5969LP
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **TI MSP430**: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430FR5969</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>47KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpmsp430fr5969` ID for `board` option in `"platformio.ini"` (Project Configuration File):

```ini
[env:lpmsp430fr5969]
platform = timsp430
board = lpmsp430fr5969
```

You can override default TI LaunchPad MSP-EXP430FR5969LP settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lpmsp430fr5969.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```ini
[env:lpmsp430fr5969]
platform = timsp430
board = lpmsp430fr5969

; change microcontroller
```

(continues on next page)
board_build.mcu = msp430fr5969
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” (Project Configuration File).

TI LaunchPad MSP-EXP430FR5969LP has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**TI LaunchPad MSP-EXP430FR5994LP**

**Contents**

- **TI LaunchPad MSP-EXP430FR5994LP**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **TI MSP430**: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.
### PlatformIO Documentation, Release 5.0.5a1

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430FR5994</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `lpmsp430fr5994` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```ini
[env:lpmsp430fr5994]
platform = timsp430
board = lpmsp430fr5994
```

You can override default TI LaunchPad MSP-EXP430FR5994LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr5994.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```ini
[env:lpmsp430fr5994]
platform = timsp430
board = lpmsp430fr5994

; change microcontroller
board_build.mcu = msp430fr5994

; change MCU frequency
board_build.f_cpu = 16000000L
```

#### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" *(Project Configuration File)*.

TI LaunchPad MSP-EXP430FR5994LP has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

TI LaunchPad MSP-EXP430FR6989LP

Contents

- TI LaunchPad MSP-EXP430FR6989LP
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **TI MSP430**: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430FR6989</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>8MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>47KB</td>
</tr>
<tr>
<td>RAM</td>
<td>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpmsp430fr6989` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:lpmsp430fr6989]
platform = timsp430
board = lpmsp430fr6989
```

You can override default TI LaunchPad MSP-EXP430FR6989LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr6989.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430fr6989]
platform = timsp430
board = lpmsp430fr6989

; change microcontroller
```

(continues on next page)
board_build.mcu = msp430fr6989
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini” *(Project Configuration File)*.

TI LaunchPad MSP-EXP430FR6989LP has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

**TI LaunchPad MSP-EXP430G2 w/ MSP430G2231**

**Contents**

- **TI LaunchPad MSP-EXP430G2 w/ MSP430G2231**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **TI MSP430**: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.
### Configuration

Please use `lpmsp430g2231` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:lpmsp430g2231]
platform = timsp430
board = lpmsp430g2231
```

You can override default TI LaunchPad MSP-EXP430G2 w/ MSP430G2231 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430g2231.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430g2231]
platform = timsp430
board = lpmsp430g2231

; change microcontroller
board_build.mcu = msp430g2231

; change MCU frequency
board_build.f_cpu = 1000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

TI LaunchPad MSP-EXP430G2 w/ MSP430G2231 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSP Debug</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

TI LaunchPad MSP-EXP430G2 w/ MSP430G2452

Contents

- TI LaunchPad MSP-EXP430G2 w/ MSP430G2452
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **TI MSP430**: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430G2452</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>256B</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

Configuration

Please use `lpmsp430g2452` ID for `board` option in “`platformio.ini` (Project Configuration File):”

```
[env:lpmsp430g2452]
platform = timsp430
board = lpmsp430g2452
```

You can override default TI LaunchPad MSP-EXP430G2 w/ MSP430G2452 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lpmsp430g2452.json`. For example, `board_build.mcu,board_build.f_cpu,etc`.

```
[env:lpmsp430g2452]
platform = timsp430
board = lpmsp430g2452

; change microcontroller
```
board_build.mcu = msp430g2452
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using debug_tool option in “platformio.ini” (Project Configuration File).

TI LaunchPad MSP-EXP430G2 w/ MSP430G2452 has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

TI LaunchPad MSP-EXP430G2553LP

Contents

- TI LaunchPad MSP-EXP430G2553LP
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform TI MSP430: MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.
Microcontroller | MSP430G2553
---|---
Frequency | 16MHz
Flash | 16KB
RAM | 512B
Vendor | TI

### Configuration

Please use `lpmsp430g2553` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:lpmsp430g2553]
platform = timsp430
board = lpmsp430g2553
```

You can override default TI LaunchPad MSP-EXP430G2553LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430g2553.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430g2553]
platform = timsp430
board = lpmsp430g2553

; change microcontroller
board_build.mcu = msp430g2553

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (Project Configuration File).

TI LaunchPad MSP-EXP430G2553LP has on-board debug probe and IS READY for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP Debug</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

1.12.32 TI TIVA

TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)

Contents

- TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform TI TIVA: Texas Instruments TM4C12x MCUs offer the industrys most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPLM4F120H5QR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

Configuration

Please use lplm4f120h5qr ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lplm4f120h5qr]
platform = titiva
board = lplm4f120h5qr
```

You can override default TI LaunchPad (Stellaris) w/ lm4f120 (80MHz) settings per build environment using board_*** option, where *** is a JSON object path from board manifest lplm4f120h5qr.json. For example, board_build.mcu, board_build.f_cpu, etc.
[env:lplm4f120h5qr]
platform = titiva
board = lplm4f120h5qr

; change microcontroller
board_build.mcu = lplm4f120h5qr

; change MCU frequency
board_build.f_cpu = 80000000L

Debugging

*Debugging* - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in "platformio.ini" (Project Configuration File).

TI LaunchPad (Stellaris) w/ lm4f120 (80MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI-ICDI</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)

- TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform **TI TIVA**: Texas Instruments TM4C12x MCUs offer the industry's most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPTM4C1230C3PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>80MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

Configuration

Please use `lptm4c1230c3pm` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:lptm4c1230c3pm]
platform = titiva
board = lptm4c1230c3pm
```

You can override default TI LaunchPad (Tiva C) w/ tm4c123 (80MHz) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lptm4c1230c3pm.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lptm4c1230c3pm]
platform = titiva
board = lptm4c1230c3pm

; change microcontroller
board_build.mcu = lptm4c1230c3pm

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging **Tools & Debug Probes** using `debug_tool` option in “platformio.ini” (Project Configuration File).

TI LaunchPad (Tiva C) w/ tm4c123 (80MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIIICDI</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

### TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)

#### Contents

- **TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **TI TIVA**: Texas Instruments TM4C12x MCUs offer the industry's most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPTM4C1294NCPDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>120MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `lptm4c1294ncpdt` ID for `board` option in `platformio.ini` (Project Configuration File):

```
[env:lptm4c1294ncpdt]
platform = titiva
board = lptm4c1294ncpdt
```

You can override default TI LaunchPad (Tiva C) w/ tm4c129 (120MHz) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `lptm4c1294ncpdt.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.
[env:lptm4c1294ncpdt]
platform = titiva
board = lptm4c1294ncpdt

; change microcontroller
board_build.mcu = lptm4c1294ncpdt

; change MCU frequency
board_build.f_cpu = 120000000L

## Debugging

**Debugging** - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “platformio.ini (Project Configuration File).

TI LaunchPad (Tiva C) w/ tm4c129 (120MHz) has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TI-ICDI</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

**1.12.33 WIZNet W7500**

**WIzwiki-W7500**

**Contents**

- [WIzwiki-W7500](#)
  - [Hardware](#)
  - [Configuration](#)
Hardware

Platform **WIZNet W7500**: The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>WIZNET7500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>48KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WIZNet</td>
</tr>
</tbody>
</table>

Configuration

Please use `wizwiki_w7500` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:wizwiki_w7500]
platform = wiznet7500
board = wizwiki_w7500
```

You can override default WIZwiki-W7500 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wizwiki_w7500.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:wizwiki_w7500]
platform = wiznet7500
board = wizwiki_w7500

; change microcontroller
board_build.mcu = wiznet7500

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

WIZwiki-W7500 supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:
[env:wizwiki_w7500]
platform = wiznet7500
board = wizwiki_w7500
upload_protocol = mbed

Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

WIZwiki-W7500 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

WIZwiki-W7500ECO

Contents

- WIZwiki-W7500ECO
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
Hardware

Platform **WIZNet W7500**: The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>WIZNET7500ECO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</td>
</tr>
<tr>
<td>RAM</td>
<td>48KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>WIZNet</td>
</tr>
</tbody>
</table>

Configuration

Please use `wizwiki_w7500eco` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:wizwiki_w7500eco]
platform = wiznet7500
board = wizwiki_w7500eco
```

You can override default WIZwiki-W7500ECO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wizwiki_w7500eco.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:wizwiki_w7500eco]
platform = wiznet7500
board = wizwiki_w7500eco

; change microcontroller
board_build.mcu = wiznet7500eco

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

WIZwiki-W7500ECO supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:wizwiki_w7500eco]
platform = wiznet7500
board = wizwiki_w7500eco

upload_protocol = mbed
```
Debugging

Debugging - “1-click” solution for debugging with a zero configuration.

**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging Tools & Debug Probes using `debug_tool` option in “platformio.ini” (Project Configuration File).

WIZwiki-W7500ECO has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

WIZwiki-W7500P

**Contents**

- WIZwiki-W7500P
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **WIZNet W7500**: The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things
### Configuration

Please use `wizwiki_w7500p` ID for `board` option in "`platformio.ini` (Project Configuration File):

```ini
[env:wizwiki_w7500p]
platform = wiznet7500
board = wizwiki_w7500p
```

You can override default WIZwiki-W7500P settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wizwiki_w7500p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:wizwiki_w7500p]
platform = wiznet7500
board = wizwiki_w7500p

; change microcontroller
board_build.mcu = wiznet7500p

; change MCU frequency
board_build.f_cpu = 48000000L
```

### Uploading

WIZwiki-W7500P supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```ini
[env:wizwiki_w7500p]
platform = wiznet7500
board = wizwiki_w7500p

upload_protocol = mbed
```

### Debugging

`Debugging` - “1-click” solution for debugging with a zero configuration.
**Warning:** You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “platformio.ini” (*Project Configuration File*).

WIZwiki-W7500P has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

<table>
<thead>
<tr>
<th>Compatible Tools</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
</tbody>
</table>

### 1.13 Custom Platform & Board

#### 1.13.1 Custom Development Platforms

*PlatformIO* can build the same binary code under different host systems via the single command `pio run` without any dependent software or requirements.

A *manifest* describes how to produce binaries for a particular platform under one or multiple host systems by a set of build scripts, toolchains, the settings for the most popular embedded boards, etc.

This guide explains how to write manifests, to support building for new development platforms.

**Step-by-Step Manual**

1. Choose *Packages* for platform
2. Create *Manifest File platform.json*
3. Create *Build Script main.py*
4. Finish with the *Installation*.

**Contents**

- *Custom Development Platforms*
  - Examples
  - Packages
  - *Manifest File platform.json*
  - *Build Script main.py*
Examples

Please take a look at the source code of existing PlatformIO Development Platforms.

Packages

Some tools are the same when compiling for several platforms, for example a common compiler. A *package* is some tool or framework that can be used when compiling for one or multiple platforms. Even if multiple platforms use the same package, the package only needs to be downloaded once. Since each package is pre-built for the different host systems (Windows, Mac, Linux), developers can get started without first compiling the tools.

PlatformIO has a registry with pre-built packages for the most popular operating systems and you can use them in your platform manifest. Custom packages can be uploaded to the PlatformIO Registry using `pio package publish` command.

Manifest File `platform.json`

Each platform definition includes a *manifest file* with a particular format that is parsed by PlatformIO when handling projects using that platform.

Here is an example `platform.json` for the fictitious platform “myplatform”:

```json
{
  "name": "myplatform",
  "title": "My Platform",
  "description": "My custom development platform",
  "homepage": "https://mycompany.com",
  "license": "Apache-2.0",
  "keywords": ["keyword_1", "keyword_N"],
  "repository": {
    "type": "git",
    "url": "https://github.com/platformio/platform-myplatform.git"
  },
  "version": "0.0.0",
  "frameworks": {
    "%FRAMEWORK_NAME_1%": {
      "package": "framework-%FRAMEWORK_NAME_1%",
      "script": "builder/frameworks/%FRAMEWORK_NAME_1%.py"
    },
    "%FRAMEWORK_NAME_N%": {
      "package": "framework-%FRAMEWORK_NAME_N%",
      "script": "builder/frameworks/%FRAMEWORK_NAME_N%.py"
    }
  },
  "packages": {
    "toolchain-gccarmnoneeabi": {
      "type": "toolchain",
      "owner": "platformio",
      "version": ">=1.40803.0,<1.40805.0"
    },
    "framework-%FRAMEWORK_NAME_1%": {
      "package": "framework-%FRAMEWORK_NAME_1%",
      "script": "builder/frameworks/%FRAMEWORK_NAME_1%.py"
    }
  }
}
```

(continues on next page)
"type": "framework",
"optional": true,
"version": "~1.10607.0"
},
"framework-%FRAMEWORK_NAME_N%": {
"type": "framework",
"optional": true,
"version": "~1.117.0"
},
"tool-direct-vcs-url": {
"type": "uploader",
"optional": true,
"version": "https://github.com/user/repo.git"
}
},
"pythonPackages": {
"pypi-pkg-1": "1.2.3",
"pypi-pkg-2": ">=2.3, <3"
}
}
}

**Build Script `main.py`**

Each platform definition must include a `main.py`.

PlatformIO’s build script is based on a next-generation build tool named SCons. PlatformIO has its own built-in firmware builder `env.BuildProgram` with deep library search. Please see the following template as start for developing your own `main.py`.

```python
"""
    Build script for test.py
test-builder.py
"""

from os.path import join
from SCons.Script import AlwaysBuild, Builder, Default, DefaultEnvironment

env = DefaultEnvironment()

# A full list with the available variables
# http://www.scons.org/doc/production/HTML/scons-user.html#app-variables
env.Replace(
    AR="ar",
    AS="gcc",
    CC="gcc",
    CXX="g++",
    OBJCOPY="objcopy",
    RANLIB="ranlib",
    
    UPLOADER=join("$PIOPACKAGES_DIR", "tool-bar", "uploader"),
    UPLOADCMD="$UPLOADER $SOURCES"
)

env.Append(
    ARFLAGS=["..."],
    
(continues on next page)```
platformio documentation, release 5.0.5a1

(continued from previous page)

```
ASFLAGS=['flag1', 'flag2', 'flagN'],
CCFLAGS=['flag1', 'flag2', 'flagN'],
CXXFLAGS=['flag1', 'flag2', 'flagN'],
LINKFLAGS=['flag1', 'flag2', 'flagN'],
CPPDEFINES=['DEFINE_1', 'DEFINE=2', 'DEFINE_N'],
LIBS=['additional', 'libs', 'here'],

BUILDERS=dict(
    ElfToBin=Builder(
        action=''.join(['$OBJCOPY', '-O', 'binary', '$SOURCES', '$TARGET']),
        suffix='.bin',
    ),
)

# The source code of "platformio-build-tool" is here
# https://github.com/platformio/platformio-core/blob/develop/platformio/builder/tools/
# →platformio.py

# # Target: Build executable and linkable firmware
# target_elf = env.BuildProgram()

# # Target: Build the .bin file
# target_bin = env.ElfToBin(join('$BUILD_DIR', 'firmware'), target_elf)

# # Target: Upload firmware
# upload = env.Alias(['upload'], target_bin, '$UPLOADCMD')
# AlwaysBuild(upload)

# # Target: Define targets
# Default(target_bin)
```

Installation

Using the “myplatform” platform example above:

1. Create a `platforms` directory in `core_dir` if it doesn’t exist.
2. Create a `myplatform` directory in `platforms`
3. Copy the `platform.json` and `builder/main.py` files to the `myplatform` directory.
4. Search the available platforms via the `pio platform search` command. You should see the new `myplatform` platform.

5. Install the `myplatform` platform via the `pio platform install` command.

Now, you can use `myplatform` as value for the `platform` option in “platformio.ini” (Project Configuration File).

Publishing

You can publish a development platform to the PlatformIO Trusted Registry using `pio package publish` command. Other developers will be able to install it. Every time when you modify a source code of a development platform you will need to increment the “version” field in “platform.json” manifest and re-publish again.

If the published development platform has an issue and you would like to remove it from the PlatformIO Trusted Registry, please use `pio package unpublish` command.

1.13.2 Custom Embedded Boards

PlatformIO has pre-built settings for many popular embedded boards. The list of these boards is available as a web page at PlatformIO Boards Explorer or through the CLI command `pio boards`.

Custom boards can also be defined from scratch or by overriding settings of existing boards. All data is declared using the JSON syntax via associative array name/value pairs.

### Contents

- Custom Embedded Boards
  - JSON Structure
  - Installation
  - Examples

### JSON Structure

The key fields are:

- **build** data is handed over to the *Development Platforms* and *Frameworks* builders
- **frameworks** is the list with supported *Frameworks*. Each working environment for each project that uses the board will choose one of the frameworks declared here.
- **platform** name of *Development Platforms*
- **upload** upload settings which depend on the **platform**

For details, see existing boards as examples, available under `.platformio/platforms/*/boards/`

```json
{
  "build": {
    "extra_flags": "-DHELLO_PLATFORMIO",
    "f_cpu": "16000000L",
    "hwids": [
      "0x1234",
      "0x5678",
      "0x9ABC"
    ]
  }
}
```

(continues on next page)
Installation

1. Create `boards` directory in `core_dir` if it doesn’t exist.
2. Create `myboard.json` file in this `boards` directory.
3. Search available boards via `pio boards` command. You should see `myboard` board.

Now, you can use `myboard` for the `board` option in “`platformio.ini`” (Project Configuration File).

Note: You can have custom boards per project. In this case, please put your board’s JSON files to `boards_dir`.

Examples

Please take a look at the source code of PlatformIO Development Platforms and navigate to `boards` folder of the repository.

1.14 Debugging

It Simply Works. Easier than ever before!
PlatformIO Debugging Solution offers a unique debugging experience for productive embedded development. Using our multi-board and multi-architecture programming experience, we simplified the debugging process in the same way. A zero debugging configuration with support for the most popular debugging probes and compatibility between IDEs and OS.

Developers can finally forget about complex UI windows which they need to pre-configure before a simple “Hello World!” debugging session. No need to know any aspects about the debugging server or how to configure it. PIO Unified Debugger does this complex work automatically having a rich configuration database per each board and debugging probe.

Just select a board, connect debugging probe (if a board does not have onboard debugging interface), specify it in PlatformIO project configuration file “platformio.ini”, and a project is ready for 1-Click debugging.

- “1-click” solution, zero configuration
- Support over 300+ embedded boards (see below)
- Multiple architectures and development platforms
- Windows, MacOS, Linux
- Built-in into PlatformIO IDE for Atom and VSCode
- Integration with Eclipse and Sublime Text

Hint: In our experience, VSCode has the best system performance, modern interface for PlatformIO Debugging Solution, and users have found it easier to get started. Key debugging features of VSCode:

- Local, Global, and Static Variable Explorer
- Conditional Breakpoints
- Expressions and Watchpoints
- Generic Registers
- Peripheral Registers
- Memory Viewer
- Disassembly
- Multi-thread support
- A hot restart of an active debugging session
1.14.1 Tutorials

- Arduino In-circuit Debugging with PlatformIO
- Use the PlatformIO Debugger on the ESP32 Using an ESP-prog
- ThingForward: First steps with PlatformIO’s Unified Debugger
- [VIDEO] ThingForward - Intro to PIO Unified Debugger using ARM mbed OS and PlatformIO IDE for VSCode
- Get started with Arduino and ESP32-DevKitC: debugging and unit testing
- Get started with ESP-IDF and ESP32-DevKitC: debugging, unit testing, project analysis
- Arduino and Nordic nRF52-DK: debugging and unit testing
- Zephyr and Nordic nRF52-DK: debugging, unit testing, project analysis
- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing
1.14.2 Configuration

PlatformIO Debugging Solution can be configured using "platformio.ini" (Project Configuration File):

1.14.3 Tools & Debug Probes

You can switch between debugging tools using debug_tool option.

**Warning:** You will need to install debug tool drivers depending on your operating system. Please check “Drivers” section for debugging tool below.

**Altera / Intel USB-Blaster**

USB Blaster Download Cable is designed for ALTERA FPGA, CPLD, Active Serial Configuration Devices and Enhanced Configuration Devices, USB 2.0 connection to the PC and JTAG, AS, PS to the target device. Official reference can be found here.
If you would like to use this tool for firmware uploading, please change upload protocol:

```python
[env:myenv]
platform = ...
board = ...
debug_tool = altera-usb-blaster
upload_protocol = altera-usb-blaster
```

More options:

- [Debugging options](#)
- [Upload options](#)

**Drivers**

Please install official drivers.

**Wiring Connections**

![Wiring Connections Diagram]

**JTAG Interface**

<table>
<thead>
<tr>
<th>USB-Blaster JTAG 10-Pin Connector</th>
<th>Board Pin</th>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TCK</td>
<td>JTAG</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>2</td>
<td>GND</td>
<td>Digital</td>
<td>Digital ground</td>
</tr>
<tr>
<td>3</td>
<td>TDO</td>
<td>Test Data Out pin</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>VCC</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TMS</td>
<td>Test Mode State pin</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>TDI</td>
<td>Test Data In pin</td>
<td></td>
</tr>
</tbody>
</table>
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigaDevice GD32V</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td>Nuclei</td>
<td>Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice GD32V SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>Nuclei</td>
<td>On-board</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
Atmel-ICE

Atmel-ICE is a powerful development tool for debugging and programming ARM® Cortex®-M based SAM and AVR microcontrollers with on-chip debug capability. Official reference can be found here.

Contents

- Configuration
- Drivers
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in `platformio.ini` (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
debg_tool = atmel-ice
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
debg_tool = atmel-ice
upload_protocol = atmel-ice
```

More options:

- Debugging options
- Upload options
Drivers

**Windows** When installing the Atmel-ICE on a computer running Microsoft Windows, the USB driver is loaded when the Atmel-ICE is first plugged in.

**Mac** Not required.

**Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel SAM</td>
<td>Atmel I SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit BLM Badge</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Crickit M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 CAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAME51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51P20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit MONSTER M4SK</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Matrix Portal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M0 Expresss</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M4 AirLift Lite</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4 Titanio</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Trellis M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pRkey</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1080KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pyBadge AirLift M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1310</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino TIAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMC21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMC21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>BrikI ABC (MBC-WB) - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Brika MBC-WB - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Minitronics v2.0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ExpLoRer</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Seeeduino Fento M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Terminal</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51P19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun 9DoF Razor IMU M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun Qwiic Micro</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD51 Thing Plus</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

avr-stub

avr-stub is a source level debugger based on GDB stub mechanism. It works with ATmega328 and Arduino Mega microcontrollers without an external programmer. The official reference can be found here.

Contents

- Configuration
- Breakpoint modes
- Debugger limitations
- Platforms
- Frameworks
- Boards

Configuration

In a nutshell, avr-stub is a piece of software (stub) that is added to your application and communicates with PlatformIO when a debug session is running. It requires several additional configuration steps in order to use it as a debug tool. To use avr-stub, the following settings in "platformio.ini" (Project Configuration File):

```ini
[env:myenv]
platform = atmelavr
board = ...
devp工具 = avr-stub
devp_port = SERIAL_PORT
```

(continues on next page)
lib_deps =
jdolinay/avr-debugger @ ~1.1

Where the value in `debug_port` is a serial port connected to your board and `jdolinay/avr-debugger` is a special library that implements the GDB stub.

In order to enable the GDB stub in your application, a call to the special function `debug_init` must be added at the beginning of your application. For example, with the Arduino framework it might look like this:

```
#include "Arduino.h"
#include "avr8-stub.h"

void setup()
{
    // initialize GDB stub
    debug_init();
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
    digitalWrite(LED_BUILTIN, HIGH);
    delay(1000);
    digitalWrite(LED_BUILTIN, LOW);
    delay(1000);
}
```

**Warning:** If your program doesn’t stop on a breakpoint, try specifying an explicit breakpoint directly in the code using the `breakpoint` function:

```
... 
void loop()
{
    breakpoint();
    digitalWrite(LED_BUILTIN, HIGH);
    delay(300);
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
}
... 
```

**Breakpoint modes**

The `avr-stub` tool supports the following three breakpoint modes:
AVR8_BREAKPOINT_MODE

<table>
<thead>
<tr>
<th>Description</th>
<th>AVR8_BREAKPOINT_MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLASH breakpoint mode, which works only with atmega328 based boards and requires a special bootloader</td>
<td>0</td>
</tr>
<tr>
<td>RAM breakpoint mode, the default mode which works out of the box with all atmega328, atmega1280, and atmega2560 based boards</td>
<td>1</td>
</tr>
<tr>
<td>FLASH breakpoint mode through Optiboot, which works with all atmega328, atmega1280, and atmega2560 based boards and requires the version 8 of the bootloader.</td>
<td>2</td>
</tr>
</tbody>
</table>

To switch between modes, specify a special macro definition `AVR8_BREAKPOINT_MODE` with the appropriate value from the table above, for example:

```
[env:myenv]
platform = atmelavr
board = uno

; Set breakpoint mode
build_flags =
  -DAVR8_BREAKPOINT_MODE=2

debug_tool = avr-stub
debug_port = SERIAL_PORT

lib_deps =
  jdolinay/avr-debugger @ ~1.1
```

**Debugger limitations**

- One external interrupt pin must be reserved for the debugger.
- Any part of your application that uses the UART module (e.g. Arduino Serial class) cannot be used in your program together with the debugger.
- When using flash breakpoints the watchdog cannot be used.

More detailed information can be found in the Important limitations of the debugger section in the official documentation.

**Platforms**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel AVR</td>
<td>Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market-and easily adapt to new ones-they are based on the industry’s most code-efficient architecture for C and assembly programming</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>
## Boards

*Note: For more detailed board information please scroll tables below by horizontal.*

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega1280</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>128KB</td>
<td>8K</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>256KB</td>
<td>8K</td>
</tr>
<tr>
<td>ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328</td>
<td>16MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>ATmega328P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2K</td>
</tr>
<tr>
<td>Adafruit Feather 328P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Alorium HiJ</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Anarduino MiniWireless</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>124KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>BQ ZUM BT-328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2K</td>
</tr>
<tr>
<td>BitWizard Raspduino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Controllino Maxi</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8K</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>FYSETC F6 V1.3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8K</td>
</tr>
<tr>
<td>LightBlue Bean</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>LowPowerLab MightyHat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2K</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2K</td>
</tr>
<tr>
<td>OpenEnergyMonitor emonPi</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2K</td>
</tr>
<tr>
<td>Prusa RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8K</td>
</tr>
<tr>
<td>RepRap RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8K</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SODAQ Moja</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Seeeduino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Digital Sandbox</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro Mini 3.3V</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SpellFoundry Sleepy Pi 2</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Talk2 Whisper Node</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TinyCircuits TinyDuino Processor Board</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ubIQio Ardhat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

**Black Magic Probe**

The Black Magic Probe is a modern, in-application debugging tool for embedded microprocessors. It is able to control and examine the state of the target microprocessor using a JTAG or Serial Wire Debugging (SWD) port and on-chip debug logic provided by the microprocessor. The probe connects to a host computer using a standard USB interface. Official reference can be found here.

Also, see Custom debugging configuration with Black Magic Probe.

**Contents**

- Configuration
- Drivers
- Wiring Connections
  - JTAG Interface
  - Serial Wire Mode Interface (SWD)
- Platforms
- Frameworks

1.14. Debugging
You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
depth_tool = blackmagic
depth_port = <CONFIGURE GDB PORT>

; Debug Port Examples
;

; Linux
depth_port = /dev/ttyACM0

; Windows for COM1-COM9
depth_port = COM3

; Windows for COM10-XXX
depth_port = \\.\COM13

; macOS
depth_port = /dev/cu.usbmodemE2C0C4C6
```

Black Magic Probe has 2 serial ports: UART and GDB. We will need “GDB” port. Please use `PlatformIO Home` > Devices or `PlatformIO Core (CLI)` and `pio device list` command to list available ports. If you do not see “Black Magic Probe GDB” port, please try both. More details.

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
depth_tool = blackmagic
depth_port = <CONFIGURE GDB PORT>

upload_port = <THE SAME AS DEBUG PORT>

; SWD interface
upload_protocol = blackmagic

; JTAG interface
upload_protocol = blackmagic-jtag
```

More options:

- Debugging options
- Upload options

Drivers

Not required.
## Wiring Connections

### JTAG Interface

<table>
<thead>
<tr>
<th>Black Magic Probe 10-Pin Connector</th>
<th>Board Pin</th>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>1</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>3</td>
<td>Digital ground</td>
</tr>
<tr>
<td>2</td>
<td>TMS</td>
<td>2</td>
<td>Test Mode State</td>
</tr>
<tr>
<td>4</td>
<td>TCLK</td>
<td>4</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>6</td>
<td>TDO</td>
<td>6</td>
<td>Test Data Out</td>
</tr>
<tr>
<td>8</td>
<td>TDI</td>
<td>8</td>
<td>Test Data In</td>
</tr>
<tr>
<td>10</td>
<td>RESET</td>
<td>10</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>

### Serial Wire Mode Interface (SWD)

<table>
<thead>
<tr>
<th>ARM 10-Pin Connector</th>
<th>ARM 20-Pin Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>VCC</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>N/U</td>
<td>N/U</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>N/U</td>
<td>N/U</td>
</tr>
<tr>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Black Magic Probe 10-Pin Connector</th>
<th>Board Pin</th>
<th>SWD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td></td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td></td>
<td>Digital ground</td>
</tr>
<tr>
<td>2</td>
<td>SWDIO</td>
<td></td>
<td>Data I/O</td>
</tr>
<tr>
<td>4</td>
<td>SWCLK</td>
<td></td>
<td>Clock</td>
</tr>
<tr>
<td>10</td>
<td>RESET</td>
<td></td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aceinna IMU</strong></td>
<td>Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.</td>
</tr>
<tr>
<td><strong>Atmel SAM</strong></td>
<td>Atmel</td>
</tr>
<tr>
<td><strong>Freescale Kinetis</strong></td>
<td>Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.</td>
</tr>
<tr>
<td><strong>Nordic nRF51</strong></td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.</td>
</tr>
<tr>
<td><strong>Nordic nRF52</strong></td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td><strong>NXP i.MX RT</strong></td>
<td>The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.</td>
</tr>
<tr>
<td><strong>NXP LPC</strong></td>
<td>The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.</td>
</tr>
<tr>
<td><strong>Silicon Labs EFM32</strong></td>
<td>Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.</td>
</tr>
<tr>
<td><strong>ST STM32</strong></td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
</tbody>
</table>
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software reuse, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>ST Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

## Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
</tr>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723IEK6</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>3D Printer controller board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
</tr>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
</tr>
</tbody>
</table>

**Note:** For more detailed board information please scroll tables below by horizontal.
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Nitrogen</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>Aceinna Low Cost RTK</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STMS2F469NIH6</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STMS2F405RG</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STMS2F405RG</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STMS2L431CB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STMS2F469IG</td>
</tr>
<tr>
<td>Aceinna OpenRTK 330L</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STMS2F469IG</td>
</tr>
<tr>
<td>Adafruit BLM Badge</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21E18A</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Adafruit Crickit M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STMS2F405RG</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21E18A</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Adafruit ItsyBitty M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Adafruit Metro M0 Expressss</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21E18A</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21E18A</td>
</tr>
<tr>
<td>Adafruit pHKey</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21E18A</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STMS2F103CBT6</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR WAN 1310</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino Nano 33 BLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STMS2F417VGT6</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STMS2F427VIT6</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STMS2F407VET6</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SARM21G18A</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SMD21G18A</td>
</tr>
<tr>
<td>Atmel SAMC21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SMC21J18A</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
</tr>
<tr>
<td>BBC micro:bit V2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52833</td>
</tr>
<tr>
<td>BL652 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>BlazDK</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Briki MBC-WB - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
</tr>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RCT6</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
</tr>
<tr>
<td>Digitstump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAMX3X8E</td>
</tr>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
</tr>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32GG9900F1024</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32LG990F256</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32WG990F256</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32ZG222F32</td>
</tr>
<tr>
<td>Econode-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
<tr>
<td>ElectronuLabs Blip</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>ElectronuLabs Papyr</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Electromith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32H750IBK6</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128V1K4</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL27Z64V1H4</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
</tbody>
</table>

1.14. Debugging
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holyiot YJ-16019</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>ItsyBitsy nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>LA76DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VG6T6</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
</tr>
<tr>
<td>MKR Sharky</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32WB55CG</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>Arm SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET</td>
</tr>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Maple (RT6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RT6</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
</tr>
<tr>
<td>Minimotronics v2.0</td>
<td>Arm SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Arm SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
</tr>
<tr>
<td>Namote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Arm SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>NXP LPC1IC24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1IC24</td>
</tr>
<tr>
<td>NXP LPC1IU34</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1IU34</td>
</tr>
<tr>
<td>NXP LPC1IU37</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1IU37</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
</tr>
<tr>
<td>NXP LPCXpresso1549</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1549</td>
</tr>
<tr>
<td>NXP i.MX RT1010 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1011DAE5A</td>
</tr>
<tr>
<td>NXP i.MX RT1015 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1015DAF5A</td>
</tr>
<tr>
<td>NXP i.MX RT1020 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1021DAG5A</td>
</tr>
<tr>
<td>NXP i.MX RT1050 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1052DVL6B</td>
</tr>
<tr>
<td>NXP i.MX RT1060 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1062DVL6A</td>
</tr>
<tr>
<td>NXP i.MX RT1064 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1064DVL6A</td>
</tr>
<tr>
<td>NXP mbed LPC1724</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Nordic Thingy::52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G071RBT6</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431KBT6</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431RBT6</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G474RET6</td>
</tr>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
</tr>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG76</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32WB55RG</td>
</tr>
<tr>
<td>PYBStick26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
</tr>
<tr>
<td>PYBstick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
</tr>
<tr>
<td>PYBstick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
</tr>
<tr>
<td>PYBstick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
</tr>
<tr>
<td>Particle Argon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Particle Boron</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Particle Xenon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
</tr>
<tr>
<td>PrntrBoard V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407RE</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L051C8T6</td>
</tr>
<tr>
<td>Raytac MDBT50Q-RX Dongle</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RG76</td>
</tr>
<tr>
<td>Ruvi Tag</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32HG322F64</td>
</tr>
<tr>
<td>SLSTK3401A Pearl Gecko PG1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32PG1B200F256GM48</td>
</tr>
<tr>
<td>SLSTK3701A Giant Gecko S1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32GG11B820F2048GL192</td>
</tr>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>SODAQ ExpLoRe</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F3348CT6</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VCT6</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
</tr>
<tr>
<td>ST 32F476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F476NGH6</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NIH6</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
</tr>
</tbody>
</table>

1.14. Debugging
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
</tr>
<tr>
<td>ST B-L475E-1OT01A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302R8T6</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410RBT6</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
</tr>
<tr>
<td>ST Nucleo F472ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F472ZET6</td>
</tr>
<tr>
<td>ST Nucleo F476ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F476ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F476ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F476ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F722ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
</tr>
<tr>
<td>ST Nucleo F722ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F746ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H743ZIT6</td>
</tr>
<tr>
<td>ST Nucleo H745ZI-Q</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H745ZIT6</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K4T6</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K6T6</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053R8T6</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073RZT6</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RET6</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KBU6</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L432KCU6</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RET6</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RGT6</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
</tr>
<tr>
<td>ST Nucleo L496ZGT-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6P</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4R5ZIT6</td>
</tr>
<tr>
<td>ST STM32F030DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
</tr>
<tr>
<td>STM32G031F6-DISCO</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G031J6</td>
</tr>
<tr>
<td>STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073VZT6</td>
</tr>
<tr>
<td>STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RB T6</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476JG</td>
</tr>
<tr>
<td>STEVAL-FCU001V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103C4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103C8T6 (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103R4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103R6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103T4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103T8T6 (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103V4 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103V6 (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103V7T6 (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103Z4 (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103Z6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103Z7 (1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103Z8 (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103Z9T6 (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103ZB (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103VC (256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103VD (384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103VF (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>STM32F401CB (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CB</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CC</td>
</tr>
<tr>
<td>STM32F401CD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CD</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CE</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RC</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RD</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RE</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410C8</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410CB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410RB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CC</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RC</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CE</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CG</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RG</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413HR</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RG</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VE</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423CH</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 516k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423RH</td>
</tr>
<tr>
<td>STM32F424RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F424RC</td>
</tr>
<tr>
<td>STM32F424RE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F424RE</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
</tr>
<tr>
<td>STM32H747I-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747IHX6</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Seeedino 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439VI</td>
</tr>
<tr>
<td>Seeeduino Fempo M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino Wio Terminal</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9ZI</td>
</tr>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>SparkFun 9DoF Razor IMU M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun Qwiic Micro</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
</tbody>
</table>
Table 52 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
</tr>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1114FN28</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072KZ</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBU6</td>
</tr>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>VAvE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
</tr>
<tr>
<td>WeAct Black Pill V2.0 (Black Pill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
</tr>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>reel_board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
</tr>
<tr>
<td>y5 LPC11U35 mbug</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
</tbody>
</table>

**CMSIS-DAP**

CMSIS-DAP is generally implemented as an on-board interface chip, providing direct USB connection from a development board to a debugger running on a host computer on one side, and over JTAG (Joint Test Action Group) or SWD (Serial Wire Debug) to the target device to access the Coresight DAP on the other. Official reference can be found [here](#).
**Configuration**

You can configure debugging tool using `debug_tool` option in `platformio.ini` *(Project Configuration File)*:

```
[env:myenv]
platform = ...
board = ...
debug_tool = cmsis-dap
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = cmsis-dap
upload_protocol = cmsis-dap
```

More options:

- *Debugging options*
- *Upload options*

**Drivers**

**Windows**  Please install *Windows serial driver* and check “USB Driver Installation” guide for your board.

**Mac**  Not required.

**Linux**  Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.
### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel SAM</td>
<td>Atmel</td>
</tr>
<tr>
<td>Freescale Kinetis</td>
<td>Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.</td>
</tr>
<tr>
<td>Maxim 32</td>
<td>Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.</td>
</tr>
<tr>
<td>Nordic nRF51</td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.</td>
</tr>
<tr>
<td>Nordic nRF52</td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td>NXP LPC</td>
<td>The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.</td>
</tr>
<tr>
<td>ST STM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
<tr>
<td>WIZ-Net W7500</td>
<td>The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things.</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>

Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168M</td>
</tr>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100M</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723IEK6</td>
<td>216M</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168M</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180M</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216M</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
<td>84M</td>
</tr>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100M</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168M</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100M</td>
</tr>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48M</td>
</tr>
</tbody>
</table>

Note: For more detailed board information please scroll tables below by horizontal.
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>Arduino Nano 33 BLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VGT6</td>
<td>168M</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168M</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168M</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>Atmel SAMC21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMC21J18A</td>
<td>48M</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48M</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48M</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>BBC micro:bit V2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52833</td>
<td>64M</td>
</tr>
<tr>
<td>BL652 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
<td>204M</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168M</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168M</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168M</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168M</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72M</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72M</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72M</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84M</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84M</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168M</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72M</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72M</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>Calliope mini</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>LPC812</td>
<td>30M</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
<td>204M</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RCT6</td>
<td>84M</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>Delta DFCM-NNN40</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48M</td>
</tr>
<tr>
<td>ElectronutLabs Blip</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>ElectronutLabs Papyr</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32H750IBK6</td>
<td>400M</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120M</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120M</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
</tbody>
</table>

1.14. Debugging

2443
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK20DX128VLH5</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK22FN512VLH12</td>
<td>120MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK66FN2M0VMID18</td>
<td>180MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL05Z32VF4M4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128VLK4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL27F64VLLH4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL43Z256VLLH4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL46Z256VLL4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKW41Z128VHT4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
</tr>
<tr>
<td>Holyiot JJ-16019</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>JsyBitsy nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>JKSoft Wallbot BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
<td>50MHz</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>MKR Sharky</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32WBS5CG</td>
<td>64MHz</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
</tr>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>MKF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>MKF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RE6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Maxim ARM mbed Enabled Development Platform for MAX32600</td>
<td>Maxim 32</td>
<td>On-board</td>
<td>MAX32600</td>
<td>24MHz</td>
</tr>
<tr>
<td>Maxim Wireless Sensor Node Demonstrator</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32610</td>
<td>24MHz</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RE6</td>
<td>100MHz</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54114J256BD64</td>
<td>100MHz</td>
</tr>
<tr>
<td>NXP mbed LPC11U24</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
</tr>
<tr>
<td>NXP mbed LPC1768</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------</td>
<td>----------</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
</tbody>
</table>

1.14. Debugging
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F103R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073RZT6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KBT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L432KCU6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L433RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476GRT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486GRT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496GJT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L496ZGP</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGTP6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo LAR5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32LAR5ZIT6</td>
<td>120MHz</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32G031J6</td>
<td>64MHz</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073VZT6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RBT6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F100RBT6</td>
<td>24MHz</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4G6FJT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>Internal</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>Internal</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>Internal</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>ST STM32</td>
<td>Internal</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>ST STM32</td>
<td>Internal</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>Internal</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
</tr>
<tr>
<td>STM32H747I-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Seeed Arch Pro</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9Z1</td>
<td>120MHz</td>
</tr>
<tr>
<td>Sparky V1 F30J</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Switch Science mbed HRM1017</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1114FN28</td>
<td>48MHz</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
</tr>
<tr>
<td>Switch Science mbed T5Y1822r2</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>ThunderFunk v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Vake v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>VNG VBLUNOS51</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>WIZwiki-W7500</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500</td>
<td>48MHz</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500ECO</td>
<td>48MHz</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500P</td>
<td>48MHz</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
</tr>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
</tbody>
</table>
### JTAG-HS1

The JTAG-HS1 programming cable is a high-speed programming solution for Xilinx® FPGAs. Official reference can be found [here](#).

### Contents

- **Configuration**
- **Platforms**
- **Frameworks**
- **Boards**

### Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” *(Project Configuration File)*:

```
[env:myenv]
platform = ...
board = ...
d.debug_tool = digilent-hs1
```

If you would like to use this tool for firmware uploading, please change upload protocol:
More options:

- **Debugging options**
- **Upload options**

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>WD-Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

### Boards

*Note:* For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
</tbody>
</table>
ESP-Prog

ESP-Prog is one of Espressif’s development and debugging tools, with functions including automatic firmware downloading, serial communication, and JTAG online debugging. ESP-Prog’s automatic firmware downloading and serial communication functions are supported on both the ESP8266 and ESP32 platforms, while the JTAG online debugging is supported only on the ESP32 platform. Official reference can be found here.

Contents

- Configuration
- Drivers
- Wiring Connections
- Tutorials
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
download_tool = esp-prog
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
download_tool = esp-prog
upload_protocol = esp-prog
```

More options:

- Debugging options
• **Upload options**

## Drivers

### Windows

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

**Mac** macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

**Linux** Please install “udev” rules 99-platformio-udev.rules. If you already installed them before, please check that your rules are up-to-date or repeat steps.

## Wiring Connections

![Wiring Connections Diagram](image)
### ESP-Prog JTAG 10-Pin Connector

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VDD Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>3</td>
<td>GND Digital ground</td>
</tr>
<tr>
<td>2</td>
<td>ESP_TMS Test Mode State</td>
</tr>
<tr>
<td>4</td>
<td>ESP_TCK JTAG Return Test Clock</td>
</tr>
<tr>
<td>6</td>
<td>ESP_TDO Test Data Out</td>
</tr>
<tr>
<td>8</td>
<td>ESP_TDI Test Data In</td>
</tr>
</tbody>
</table>

### Tutorials

- Use the PlatformIO Debugger on the ESP32 Using an ESP-prog

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espresif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

### Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32v IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IoTaaP Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos DI MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
FTDI Chip

FTDI Chip develops innovative silicon solutions that enhance interaction with today’s technology. When a designer needs to add a USB port, rest assured that FTDI Chip has a full range of USB solutions to get the job done. Official reference can be found here.

Contents

- Configuration
- Drivers
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = ftdi
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = ftdi
upload_protocol = ftdi
```

More options:

- Debugging options
- Upload options
Drivers

Windows

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

Mac  macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

Linux  Please install “udev” rules 99-platformio-udev.rules. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>RISC-V GAP</td>
<td>GreenWaves GAP8 IoT application processor enables the cost-effective development, deployment and autonomous operation of intelligent sensing devices that capture, analyze, classify and act on the fusion of rich data sources such as images, sounds or vibrations.</td>
</tr>
<tr>
<td>Shakti</td>
<td>Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP-IDF</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the “things” in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>PULP OS</td>
<td>PULP is a silicon-proven Parallel Ultra Low Power platform targeting high energy efficiencies. The platform is organized in clusters of RISC-V cores that share a tightly-coupled data memory</td>
</tr>
<tr>
<td>Shakti SDK</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

## Boards

Note: For more detailed board information please scroll tables below by horizontal.
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Arty A7-100: Arty-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>GAPuino GAP8</td>
<td>RISC-V GAP</td>
<td>On-board</td>
<td>GAP8</td>
<td>250MHz</td>
<td>64MB</td>
<td>8MB</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>Parasha on Artix-7 100T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Pinaka on Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Vajra on Arty A7-100: Arty-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
</tbody>
</table>

GD-LINK

GD-Link adapter is a three-in-one multi-function development tool for GD32 series of MCUs. It provides CMSIS-DAP debugger port with JTAG/SWD interface. Official reference can be found [here](#).

**Contents**

- Configuration
- Drivers
- Wiring Connections
  - JTAG Interface
  - Serial Wire Mode Interface (SWD)
Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = gd-link
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = gd-link
upload_protocol = gd-link
```

More options:

- **Debugging options**
- **Upload options**

Drivers

- **Windows** Check vendor recommendations.
- **Mac** Not required.
- **Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Wiring Connections

**JTAG Interface**

<table>
<thead>
<tr>
<th>GD-Link Connector</th>
<th>JTAG 20-Pin Connector</th>
<th>Board Pin</th>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+3V3</td>
<td>VCC</td>
<td>VCC</td>
<td>VCC</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>TMS/IO</td>
<td>TMS</td>
<td>TMS</td>
<td>TMS</td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>TCK/CLK</td>
<td>TCK</td>
<td>TCK</td>
<td>TCK</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>TDO/SWO</td>
<td>TDO</td>
<td>TDO</td>
<td>TDO</td>
<td>Test Data Out pin</td>
</tr>
<tr>
<td>TDI</td>
<td>TDI</td>
<td>TDI</td>
<td>TDI</td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>GDN</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>TReset</td>
<td>RESET</td>
<td>RESET</td>
<td>RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>
Serial Wire Mode Interface (SWD)

<table>
<thead>
<tr>
<th>GD-Link SWD 20-Pin Connector Pin</th>
<th>Board SWD Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+3V3</td>
<td>VCC</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>TMS/IO</td>
<td>SWDIO</td>
<td>Data I/O</td>
</tr>
<tr>
<td>TCK/CLK</td>
<td>SWCLK</td>
<td>Clock</td>
</tr>
<tr>
<td>TReset</td>
<td>RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigaDevice GD32V</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td>Nuclei</td>
<td>Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice GD32V SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.
### oddWires IOT-Bus JTAG

This IoT-Bus module provides JTAG debugging for the oddWires IoT-Bus Io and oddWires IoT-Bus Proteus boards (can be used with other boards too, see wiring connections below). The board uses the FT232H to provide a USB controller with JTAG support. Both debugging and flashing is possible using this port. Official reference can be found here.
Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```yaml
[env:myenv]
platform = ...
board = ...
debug_tool = iot-bus-jtag
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```yaml
[env:myenv]
platform = ...
board = ...
debug_tool = iot-bus-jtag
upload_protocol = iot-bus-jtag
```

More options:

- [Debugging options](#)
- [Upload options](#)

Drivers

**Windows**

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

**Mac** macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

**Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Wiring Connections

<table>
<thead>
<tr>
<th>IOT-Bus Pin</th>
<th>JTAG Pin</th>
<th>Board Pin</th>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3V3</td>
<td>VCC</td>
<td>VCC</td>
<td>JTAG</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>JTAG</td>
<td>Digital ground</td>
</tr>
<tr>
<td>12</td>
<td>TDI</td>
<td>Test Data In pin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>TMS</td>
<td>Test Mode State pin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>TCK</td>
<td>JTAG Return Test Clock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>TDO</td>
<td>Test Data Out pin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EN</td>
<td>RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

## Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-dino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32wn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

*Note: For more detailed board information please scroll tables below by horizontal.*
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IoTaaS Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>1.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MFI MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MIni ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
J-LINK

SEGGER J-Links are the most widely used line of debug probes available today. They’ve proven their value for more than 10 years with over 400,000 units sold, including OEM versions and on-board solutions. This popularity stems from the unparalleled performance, extensive feature set, large number of supported CPUs, and compatibility with all popular development environments. Official reference can be found here.

- J-Link Supported Devices

Also, see Custom debugging configuration with J-Link GDB Server.

Contents

- Configuration
  - Custom Server
- Drivers
- Wiring Connections
  - JTAG Interface
  - Serial Wire Mode Interface (SWD)
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using debug_tool option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = jlink
```

If you would like to use this tool for firmware uploading, please change upload protocol:
```yaml
[env:myenv]
platform = ...
board = ...
debug_tool = jlink

; SWD interface
upload_protocol = jlink

; JTAG interface
upload_protocol = jlink-jtag

More options:

- **Debugging options**
- **Upload options**

**Custom Server**

```yaml
[env:debug_jlink]
platform = ststm32
framework = mbed
board = nucleo_f446re
debug_tool = jlink
debug_port = :2331
debug_server = /full/path/to/JLinkGDBServerCL
-singlerun
-if
SWD
-select
USB
-port
2331
-device
STM32F446RE
```

**Drivers**

**Windows**

1. Start debugging session using *PlatformIO IDE*. PlatformIO will install J-Link software dependencies
2. Navigate to `core_dir/packages/tool-jlink/USBDriver`
3. Run `InstDrivers.exe`

**Mac**  Not required.

**Linux**  Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.
### Wiring Connections

#### JTAG Interface

<table>
<thead>
<tr>
<th>J-Link JTAG 20-Pin Connector</th>
<th>Board JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>5</td>
<td>TDI</td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>7</td>
<td>TMS</td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>9</td>
<td>TCK</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>13</td>
<td>TDO</td>
<td>Test Data Out pin</td>
</tr>
<tr>
<td>15</td>
<td>RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)</td>
</tr>
</tbody>
</table>

#### Serial Wire Mode Interface (SWD)

<table>
<thead>
<tr>
<th>J-Link SWD 20-Pin Connector</th>
<th>Board SWD Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>7</td>
<td>SWDIO</td>
<td>Data I/O</td>
</tr>
<tr>
<td>9</td>
<td>SWCLK</td>
<td>Clock</td>
</tr>
<tr>
<td>15</td>
<td>RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aceinna IMU</td>
<td>Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.</td>
</tr>
<tr>
<td>Atmel SAM</td>
<td>Atmel SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.</td>
</tr>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Freescale Kinetis</td>
<td>Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.</td>
</tr>
<tr>
<td>GigaDevice GD32V</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td>Infineon XMC</td>
<td>Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>Maxim 32</td>
<td>Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.</td>
</tr>
<tr>
<td>Nordic nRF51</td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.</td>
</tr>
<tr>
<td>Nordic nRF52</td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td>Nuclei</td>
<td>Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.</td>
</tr>
<tr>
<td>NXP i.MX RT</td>
<td>The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.</td>
</tr>
<tr>
<td>NXP LPC</td>
<td>The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.</td>
</tr>
<tr>
<td>Shakti</td>
<td>Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
<tr>
<td>Silicon Labs EFM32 Gecko</td>
<td>Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory configurations up to 256 kB, 32 kB of RAM and CPU speeds up to 48 MHz. Based on the powerful ARM Cortex-M core, the Gecko family features innovative low energy techniques, short wake-up time from energy saving modes and a wide selection of peripherals, making it ideal for battery operated applications and other systems requiring high performance and low-energy consumption.</td>
</tr>
<tr>
<td>STM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
<tr>
<td>Teensy</td>
<td>Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.</td>
</tr>
</tbody>
</table>
# Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td>ESPressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>GigaDevice GD32VF103 SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td>Shakti SDK</td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopencm3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
### Platforms

**Boards**

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
</tr>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
</tr>
<tr>
<td>32F723EDISCOVERI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723IEK6</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769VIT6</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
</tr>
<tr>
<td>96Boards Argonkey (ST-VAL-MKI187V1)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
</tr>
<tr>
<td>96Boards Nitrogen</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Aceinna Low Cost RTK</td>
<td>Aceinna IMU</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32L431CB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F469IG</td>
</tr>
<tr>
<td>Aceinna OpenRTK330L</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F469IG</td>
</tr>
<tr>
<td>Adafruit BLM Badge</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Adafruit Crickit M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Adafruit Feather M4 CAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51P20A</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Adafruit Hallowing M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
</tr>
<tr>
<td>Adafruit MONSTER M4SK</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
</tr>
<tr>
<td>Adafruit Matrix Portal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit Metro M0 Expresss</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit Metro M4 AirLift Lite</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
</tr>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
</tr>
<tr>
<td>Adafruit PyPortal M4 Titano</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
</tr>
<tr>
<td>Adafruit Trillis M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
</tr>
<tr>
<td>Adafruit pIRkey</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
</tr>
<tr>
<td>Adafruit pyBadge AirLift M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR WAN 1310</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino Nano 33 BLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F427VIT6</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
</tr>
<tr>
<td>Arty A7-100: Artix-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Atmel SAMC21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMC21J18A</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>BBC micro:bit V2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52833</td>
</tr>
<tr>
<td>BL652 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
</tr>
</tbody>
</table>

**Table 56 – continued from previous page**
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52832</td>
</tr>
<tr>
<td>BluDK</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>nRF51822</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Briki MBC-WB - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
</tr>
<tr>
<td>Calliope mini</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>nRF51822</td>
</tr>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1347</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RCT6</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
<tr>
<td>D-datao-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>nRF52832</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>DipCortex M3</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1347</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
</tr>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32GG990F1024</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32LG990F256</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32WG990F256</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32ZG222F32</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>ESP32vn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Econode-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
</tr>
<tr>
<td>Electronut Labs Blip</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>nRF52840</td>
</tr>
<tr>
<td>Electronut Labs Papyr</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>nRF52840</td>
</tr>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32H750IBK6</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>Expressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
</tr>
<tr>
<td>Expressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------</td>
<td>--------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGTT6</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK20DX128VLL5</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK22FN512VLLH12</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK66FN2M0VMD18</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK82FN256VLL15</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL05Z32VFM4</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128VLK4</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL27Z64VLLH4</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL43Z256VLLH4</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL46Z256VLL4</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL82Z</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKL82Z128VLLK7</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKW24D512</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKW412512VHT4</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>Nuclei</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>Nuclei</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>ST STM32</td>
<td></td>
<td>STM2L082CZY6</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td></td>
<td>STM2L082CZY6</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetis</td>
<td></td>
<td>MK64FN1M0VDC12</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
</tr>
<tr>
<td>Holyiot JY-16019</td>
<td>Nordic nRF52</td>
<td></td>
<td>NRF52832</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoTaaP Magnolia</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ItsyBitsy nRF52840 Express</td>
<td>Nordic nRF52</td>
<td></td>
<td>NRF52840</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM2L476VGT6</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
</tr>
<tr>
<td>LPCXpresso0824-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100DVK</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td></td>
<td>STM2F070CBT6</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td></td>
<td>STM2F070CBT6</td>
</tr>
<tr>
<td>MAX32620FTHR</td>
<td>Maxim 32</td>
<td></td>
<td>MAX32620FTHR</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKit</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MKR Sharky</td>
<td>ST STM32</td>
<td></td>
<td>STM32WB55CG</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Malyan M200 VI</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32620</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F339ZIY6</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZG6T6</td>
</tr>
<tr>
<td>Mintronics v2.0</td>
<td>Amel SAM</td>
<td>External</td>
<td>SAMD211J18A</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Amel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG6T6</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Amel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>NXP LPC11C24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11C24</td>
</tr>
<tr>
<td>NXP LPC11U34</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U34</td>
</tr>
<tr>
<td>NXP LPC11U37</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U37</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
</tr>
<tr>
<td>NXP LPCxpresso1549</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1549</td>
</tr>
<tr>
<td>NXP LPCxpresso3414</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC5411J256BD64</td>
</tr>
<tr>
<td>NXP LPCxpresso34608</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54608ET6T12</td>
</tr>
<tr>
<td>NXP LPCxpresso55S16</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC55S16</td>
</tr>
<tr>
<td>NXP LPCxpresso55S69</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC55S69</td>
</tr>
<tr>
<td>NXP i.MX RT1010 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1011DAE5A</td>
</tr>
<tr>
<td>NXP i.MX RT1015 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1015DAF5A</td>
</tr>
<tr>
<td>NXP i.MX RT1020 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1021DAF5A</td>
</tr>
<tr>
<td>NXP i.MX RT1050 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1052DVL6B</td>
</tr>
<tr>
<td>NXP i.MX RT1060 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1062DVL6A</td>
</tr>
<tr>
<td>NXP i.MX RT1064 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1064DVL6A</td>
</tr>
<tr>
<td>NXP mbed LPC11U24</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>NXP mbed LPC1768</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Nordic Thingy.:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G071RBT6</td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431KB76</td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431KB76</td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G474RET6</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------------</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Olimexino-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB7T6</td>
</tr>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB7T6</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG7T6</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32WB55RG</td>
</tr>
<tr>
<td>PYBstick26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
</tr>
<tr>
<td>PYBstick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F142RE</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
</tr>
<tr>
<td>Parashu on Artix-7 100T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
</tr>
<tr>
<td>Particle Argon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Particle Boron</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Particle Xenon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
</tr>
<tr>
<td>Pinaka on Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
</tr>
<tr>
<td>PrntrBoard V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407RE</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RB7T6</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RB7T6</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L051C8T6</td>
</tr>
<tr>
<td>Raytac MDBT30Q-RX Dongle</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>RedBearLab BLE Blend 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>RedBearLab BLE N51822</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RG7T6</td>
</tr>
<tr>
<td>Ruvi Tag</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>SEGGER IP Switch Board</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK66F62M0VMD18</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32HG32F64</td>
</tr>
<tr>
<td>SLSTK3401A Pearl Gecko PG1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32PG1B200F256GM48</td>
</tr>
<tr>
<td>SLSTK3701A Giant Gecko S1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32GG1B820F2048OL192</td>
</tr>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>SODAQ ExpLoRer</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VCT6</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZH7T6</td>
</tr>
<tr>
<td>ST 32F429DIDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
</tr>
<tr>
<td>ST 32F469DIDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>ST 32F769DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NIH6</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
</tr>
<tr>
<td>B-L475E-IOT01A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VG6T</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302R8T6</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RE6T</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RE6T</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410RBT6</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RE6T</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZH6T</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F756ZG</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
</tr>
<tr>
<td>ST Nucleo H743ZI-Q</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H743ZIT6</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K4T6</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K6T6</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053R8T6</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073RZ</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RE6T</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KBU6</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L432KCU6</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RE6T</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RG6T</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RG6T</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZG6T</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZG6T</td>
</tr>
</tbody>
</table>

Table 56 – continued from previous page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo LARSZI</td>
<td>ST STM32</td>
<td>ST STM32</td>
<td>ST STM32L4RSZIT6</td>
</tr>
<tr>
<td>ST STM32F030SDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F030RBT6</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F051RRT6</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F303VCT6</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F407VGT6</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32G031J6</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L073VZT6</td>
</tr>
<tr>
<td>ST STM32L152RBT6</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32L152RBT6</td>
</tr>
<tr>
<td>ST STM32L476JG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L476JG</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F401CCU6</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F407ZGT6</td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F107VCT6</td>
</tr>
<tr>
<td>STM3220C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F373VCT6</td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F072VB6</td>
</tr>
<tr>
<td>STM32F030C4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103C4</td>
</tr>
<tr>
<td>STM32F030C6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103C6</td>
</tr>
<tr>
<td>STM32F030C8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103C8T6</td>
</tr>
<tr>
<td>STM32F030CB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103CBT6</td>
</tr>
<tr>
<td>STM32F030R4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103R4</td>
</tr>
<tr>
<td>STM32F030R6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103R6</td>
</tr>
<tr>
<td>STM32F030R8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103R8T6</td>
</tr>
<tr>
<td>STM32F030RB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RBT6</td>
</tr>
<tr>
<td>STM32F030RC (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RCT6</td>
</tr>
<tr>
<td>STM32F030RD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RD</td>
</tr>
<tr>
<td>STM32F030RE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RET6</td>
</tr>
<tr>
<td>STM32F030RF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RF</td>
</tr>
<tr>
<td>STM32F030RG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RG</td>
</tr>
<tr>
<td>STM32F030T4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103T4</td>
</tr>
<tr>
<td>STM32F030T6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103T6</td>
</tr>
<tr>
<td>STM32F030T8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103T8T6</td>
</tr>
<tr>
<td>STM32F030TB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103TB6</td>
</tr>
<tr>
<td>STM32F030V8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103V8</td>
</tr>
<tr>
<td>STM32F030VB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VB6</td>
</tr>
<tr>
<td>STM32F030VC (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VC76</td>
</tr>
<tr>
<td>STM32F030VD (48k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VD6</td>
</tr>
<tr>
<td>STM32F030VE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VE6</td>
</tr>
<tr>
<td>STM32F030VF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VF</td>
</tr>
<tr>
<td>STM32F030VG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VG</td>
</tr>
<tr>
<td>STM32F030ZG (96k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZCT6</td>
</tr>
<tr>
<td>STM32F030ZD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZDT6</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZET6</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZF</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZG</td>
</tr>
<tr>
<td>STM32F030CB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F303C6T6</td>
</tr>
<tr>
<td>STM32F401CC (64k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F401CC</td>
</tr>
<tr>
<td>STM32F401CD (96k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F401CD</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
<td>-------</td>
<td>--------------</td>
</tr>
<tr>
<td>STM32F401CE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CE</td>
</tr>
<tr>
<td>STM32F401RB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RB</td>
</tr>
<tr>
<td>STM32F401RC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RC</td>
</tr>
<tr>
<td>STM32F401RD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RD</td>
</tr>
<tr>
<td>STM32F401RE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RE</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
</tr>
<tr>
<td>STM32F410C8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410C8</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410CB</td>
</tr>
<tr>
<td>STM32F410R8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410R8</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CC</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RC</td>
</tr>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CE</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RG</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CG</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RG</td>
</tr>
<tr>
<td>STM32F413RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RH</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RG</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VE</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423CH</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423RH</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RC</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RE</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
</tr>
<tr>
<td>STM32H747I-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747XI6H</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>ST STM32</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>ST STM32</td>
<td>External</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Seeedino 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439VI</td>
</tr>
<tr>
<td>Seeeduino Femto M0</td>
<td>ST STM32</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>ST STM32</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>ST STM32</td>
<td>External</td>
<td>SAMD51P19A</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>ST STM32</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>ST STM32</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9ZI</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>ST STM32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103C8T6</td>
</tr>
<tr>
<td>Sipeed MAIX Bit</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
</tr>
<tr>
<td>Sipeed MAIX Bit with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
</tr>
<tr>
<td>SparkFun 9DoF Razor IMU M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>SparkFun Qwiic Micro</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
</tr>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>SparkFun SAMD51 Thing Plus</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
</tr>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1114FN28</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>Teensy</td>
<td>External</td>
<td>MK20DX256</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>Teensy</td>
<td>External</td>
<td>MK64FX512</td>
</tr>
<tr>
<td>Teensy 3.6</td>
<td>Teensy</td>
<td>External</td>
<td>MK66FX1M0</td>
</tr>
<tr>
<td>Teensy 4.0</td>
<td>Teensy</td>
<td>External</td>
<td>IMXRT1062</td>
</tr>
<tr>
<td>Teensy 4.1</td>
<td>Teensy</td>
<td>External</td>
<td>IMXRT1062</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>Teensy</td>
<td>External</td>
<td>MKL26Z64</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072KZ</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F41CEU6</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F03TB6</td>
</tr>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>SAM32F446RET6</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>Vajra on Arty A7-100: Artix-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
</tr>
<tr>
<td>VintLabs ESP32 DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
</tbody>
</table>

Table 56 – continued from previous page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZwiki-W7500</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500ECO</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500P</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
</tr>
<tr>
<td>WeMos DJ MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
</tr>
<tr>
<td>XMC1100 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1400</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4200</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4700</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>decawave DWI001 Module Development Board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
</tr>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
</tr>
<tr>
<td>oddWires IoT-Bus Jo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
</tr>
<tr>
<td>reel_board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F437VG</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZY6</td>
</tr>
<tr>
<td>y3 LPC11U35 mbug</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
</tr>
<tr>
<td>y3 nRF51822 mbit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
</tr>
</tbody>
</table>
Mini-Module FT2232H

The FT2232H Mini Module is a USB to dual channel serial/MPSS/FIFO interface converter module based on the FT2232H USB Hi-Speed IC. The FT2232H handles all the USB signalling and protocol handling. The module provides access to device I/O interfaces via 2 double row 0.1” pitch male connectors. The module is ideal for development purposes to quickly prove functionality of adding USB to a target design. Official reference can be found here

Contents

- Configuration
- Drivers
- Wiring Connections
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = minimodule
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = minimodule
upload_protocol = minimodule
```

More options:
PlatformIO Documentation, Release 5.0.5a1

- Debugging options
- Upload options

Drivers

Windows

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

Mac macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

Linux Please install “udev” rules 99-platformio-udev.rules. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Wiring Connections

<table>
<thead>
<tr>
<th>FT2232H Mini-Module Pin</th>
<th>Board Pin</th>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>GND</td>
<td></td>
<td>Digital ground</td>
</tr>
<tr>
<td>AD0</td>
<td>TCK</td>
<td></td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>AD1</td>
<td>TDI</td>
<td></td>
<td>Test Data In</td>
</tr>
<tr>
<td>AD2</td>
<td>TDO</td>
<td></td>
<td>Test Data Out</td>
</tr>
<tr>
<td>AD3</td>
<td>TMS</td>
<td></td>
<td>Test Mode State</td>
</tr>
<tr>
<td>RESET#</td>
<td>RESET</td>
<td></td>
<td>Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)</td>
</tr>
</tbody>
</table>

You will also need to connect Vbus [CN3-1] to Vcc [CN3-3] of FT2232H Mini-Module to power the FTDI chip. See FT2232H Mini-Module Datasheet

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espresif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

## Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Arti FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-daino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32vn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPimo32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IoTaaS Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy v4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MIni ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

**MSP Debug**

The MSP debug stack (MSPDS) for all MSP430™ microcontrollers (MCUs) and SimpleLink™ MSP432™ devices consists of a static library on the host system side as well as an embedded firmware that runs on debug tools including the MSP-FET, MSP-FET430UIF or on-board eZ debuggers. It is the bridging element between all PC software and all MSP430 and SimpleLink MSP432 microcontroller derivatives and handles tasks such as code download, stepping through code or break points. Official reference can be found [here](#).
Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = mspdebug
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = mspdebug
upload_protocol = mspdebug
```

More options:

- **Debugging options**
- **Upload options**

Drivers

**Windows** Please “USB Driver Installation” guide for your board.

**Mac** Not required.

**Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TI MSP430</strong></td>
<td>MSP430 microcontrollers (MCUs) from Texas Instruments (TI) are 16-bit, RISC-based, mixed-signal processors designed for ultra-low power. These MCUs offer the lowest power consumption and the perfect mix of integrated peripherals for thousands of applications.</td>
</tr>
</tbody>
</table>
### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
</tbody>
</table>

### Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI FraunchPad MSP-EXP430FR5739LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>15.37KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430F5529</td>
<td>25MHz</td>
<td>47KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2311LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR2311</td>
<td>16MHz</td>
<td>3.75KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2433LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR2433</td>
<td>8MHz</td>
<td>15KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR4133LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR4133</td>
<td>8MHz</td>
<td>15KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5969LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5994LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5994</td>
<td>16MHz</td>
<td>256KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2231</td>
<td>1MHz</td>
<td>2KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2452</td>
<td>16MHz</td>
<td>8KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2553LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2553</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
</tbody>
</table>
Olimex ARM-USB-OCD-H

High-speed 3-IN-1 fast USB ARM/ESP32 JTAG, USB-to-RS232 virtual port and power supply 5VDC device. Official reference can be found here.

Contents

- Configuration
- Drivers
- Wiring Connections
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using debug_tool option in "platformio.ini" (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-oecd-h
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-oecd-h
upload_protocol = olimex-arm-usb-oecd-h
```

More options:

- Debugging options
- Upload options
Drivers

Windows

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

Mac macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

Linux Please install “udev” rules 99-platformio-udev.rules. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Wiring Connections

<table>
<thead>
<tr>
<th>Olimex ARM-USB-OCD-H JTAG 20-Pin Connector</th>
<th>Board JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC</td>
<td>1</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>GND</td>
<td>4</td>
<td>Digital ground</td>
</tr>
<tr>
<td>TDI</td>
<td>5</td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>TMS</td>
<td>7</td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>TCK</td>
<td>9</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>TDO</td>
<td>13</td>
<td>Test Data Out pin</td>
</tr>
<tr>
<td>RESET</td>
<td>3</td>
<td>Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)</td>
</tr>
</tbody>
</table>

| VCC (optional)                           | 2             | |
| TRST                                     | 3             | |
| 4                                         | N/U 3          | |
| 5                                         | N/U 5          | |
| 6                                         | GND            | |
| 8                                         | GND            | |
| TDI                                       | 5             | |
| TMS                                       | 7             | |
| TCK                                       | 9             | |
| TDO                                       | 13            | |
| SWD                                       | 1             | |
| 2                                         | GND            | |
| SWO                                      | 7             | |
| SWCLK                                     | 9             | |
| SWO                                      | 13            | |
| 16                                        | GND            | |
| 18                                        | GND            | |
| 20                                        | GND            | |
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>WD-Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>ESP32</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-daino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32vn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPin32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espresif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espresif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetele-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Iotaesp Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Olimex ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Olimex ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Olimex ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>ESP32</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Speed MAIX Bit</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Speed MAIX Bit with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Speed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Speed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Speed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Speed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espresif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espresif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espresif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espresif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espresif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Wemos Lolin D32</td>
<td>Espresif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Wemos Lolin D32 PRO</td>
<td>Espresif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 58 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-BusIo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-BusProteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Olimex ARM-USB-OCD

3-IN-1 fast USB ARM/ESP32 JTAG, USB-to-RS232 virtual port and power supply 5-9-12VDC device (supported by OpenOCD ARM debugger software). Official reference can be found [here](#).

Contents

- Configuration
- Drivers
- Wiring Connections
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-ocd
```

If you would like to use this tool for firmware uploading, please change upload protocol:
More options:

- **Debugging options**
- **Upload options**

### Drivers

**Windows**

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

**Mac** macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

**Linux** Please install “udev” rules 99-platformio-udev.rules. If you already installed them before, please check that your rules are up-to-date or repeat steps.

### Wiring Connections

![Wiring Diagram](image)
### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP-IDF</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>WD-Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

---

**Olimex ARM-USB-OCD JTAG 20-Pin Connector**

<table>
<thead>
<tr>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC  Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4</td>
<td>GND  Digital ground</td>
</tr>
<tr>
<td>5</td>
<td>TDI  Test Data In pin</td>
</tr>
<tr>
<td>7</td>
<td>TMS  Test Mode State pin</td>
</tr>
<tr>
<td>9</td>
<td>TCK  JTAG Return Test Clock</td>
</tr>
<tr>
<td>13</td>
<td>TDO  Test Data Out pin</td>
</tr>
<tr>
<td>3</td>
<td>RESET  Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)</td>
</tr>
</tbody>
</table>

---

1.14. Debugging
## Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32yn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IoTaaP Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>ESP32</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX G0</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MFI MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 59 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Olimex ARM-USB-TINY-H

Low-cost and high-speed ARM/ESP32 USB JTAG. Official reference can be found here.

Contents

- Configuration
- Drivers
- Wiring Connections
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):
If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-tiny-h

upload_protocol = olimex-arm-usb-tiny-h
```

More options:

- *Debugging options*
- *Upload options*

**Drivers**

**Windows**

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

**Mac** macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

**Linux** Please install “udev” rules *99-platformio-udev.rules*. If you already installed them before, please check that your rules are up-to-date or repeat steps.

**Wiring Connections**

![Wiring Diagram]
### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP-IDF</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>WD-Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>
Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32yn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IotaaP Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OILMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OILMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OILMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>RVjgpa: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td></td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MFI MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
Table 60 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS Lolin D32</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS Lolin D32 PRO</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS Lolin32</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Olimex ARM-USB-TINY

Low-cost and high-speed ARM/ESP32 USB JTAG. Official reference can be found here.

Contents

- Configuration
- Drivers
- Wiring Connections
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (Project Configuration File):
If you would like to use this tool for firmware uploading, please change upload protocol:

```plaintext
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-jtag-tiny
upload_protocol = olimex-jtag-tiny
```

More options:
- Debugging options
- Upload options

**Drivers**

**Windows**
- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

**Mac** macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

**Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.

**Wiring Connections**

![Wiring Diagram](image-url)
Olimex ARM-USB-TINY 20-Pin Connector | Board JTAG Pin | Description
--- | --- | ---
1 | VCC | Positive Supply Voltage — Power supply for JTAG interface drivers
4 | GND | Digital ground
5 | TDI | Test Data In pin
7 | TMS | Test Mode State pin
9 | TCK | JTAG Return Test Clock
13 | TDO | Test Data Out pin
3 | RESET | Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Espressif IoT Development Framework</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>WD-Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>
## Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>DOT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32vln IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IoTaaP Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>RVhpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX Bit with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MFI MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Continued on next page
QEMU

QEMU is a free and open-source emulator that performs hardware virtualization. Official reference can be found [here](#).

## Contents

- **Configuration**
- **Platforms**
- **Frameworks**
- **Boards**

## Configuration

You can configure debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = qemu
```

More options:

- **Debugging options**
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Renode

Renode is a development framework which accelerates IoT and embedded systems development by letting you simulate physical hardware systems - including both the CPU, peripherals, sensors, environment and wired or wireless medium between nodes. For more information, see Renode’s official website.

Contents

- Configuration
- Installation
- Custom Settings
  - Additional Analyzers
  - Redirecting peripherals output
- Examples
### Configuration

You can configure Renode as a debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = renode
```

More options:

- [Debugging options](#)

### Installation

We will automatically install for you the latest Renode package using PlatformIO package manager. The only requirement is to install Mono/.NET framework.

- **Windows**  On Windows 7, download and install .NET Framework 4.7. Windows 10 ships with .NET by default, so no action is required there.
- **Mac**    Install Homebrew and the `mono` package using `brew install mono`.
- **Linux**  Install the `mono-complete` package as per the installation instructions for various Linux distributions which can be found on the Mono project website.

Check the [official Renode installation guide](#) for more details.

### Custom Settings

If the default Renode configuration is not suitable for your project, it’s possible to override the default Renode flags in "platformio.ini" (Project Configuration File). The following sections describe typical use cases where custom settings might be useful.

### Additional Analyzers

Additional analyzers might be handy in cases when an application prints output to a peripheral which is not visible by default. For example, to open an additional UART window while using Renode as the upload tool, a special command `showAnalyzer` should be added to `upload_flags` option, e.g.:

```ini
[env:hifive1-revb]
platform = sifive
framework = zephyr
board = hifive1-revb
; Override the default upload settings
upload_command = renode $UPLOAD_FLAGS
```

(continues on next page)
upload_flags =
    -e include @scripts/single-node/sifive_fe310.resc
    -e showAnalyzer uart1
    -e sysbus LoadELF @$SOURCE
    -e start

Redirecting peripherals output

It might be useful to redirect output from device peripherals to a more convenient communication channel. A typical example is redirecting UART output to a socket which can be opened by `pio device monitor` while using Renode as the debug tool. The default debug flags can be overridden using the `debug_server` option. For example, the following configuration can be used to redirect output from the UART0 port on the hifive1-revb board:

```plaintext
[env:hifive1-revb]
platform = sifive
framework = zephyr
board = hifive1-revb
; Override the default debug settings
debug_tool = custom
debug_port = localhost:3333
debug_server = renode
  --hide-log
  -e machine StartGdbServer 3333 True
  -e emulation CreateServerSocketTerminal 4321 "externalUART" false
  -e connector Connect uart0 externalUART
debug_extra_cmds =
    monitor start

# Monitor port for Renode integration
monitor_port = socket://localhost:4321
```

Examples

- Pre-configured projects for Zephyr RTOS, FreeRTOS, and Arduino by Carlos Eduardo

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

RV-LINK

RISC-V emulator implemented with RISC-V development board. Unlike other emulators: RV-LINK interacts directly with GDB via a USB serial port and does not require an intermediary such as OpenOCD. Official reference can be found here.

Contents

- Configuration
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using debug_tool option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
depth  _tool = rv-link
```
If you would like to use this tool for firmware uploading, please change upload protocol:

```plaintext
[env:myenv]
platform = ...
board = ...
debug_tool = rv-link
upload_protocol = rv-link
```

More options:

- Debugging options
- Upload options

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigaDevice GD32V</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td>Nuclei SDK</td>
<td>Find professional RISC-V Processor IP in Nuclei, first professional RISC-V IP company in Mainland China, match all your requirements in AIoT Era.</td>
</tr>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice GD32VF103 SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
<tr>
<td>Nuclei SDK</td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
</tbody>
</table>

### Boards

**Note:** For more detailed board information please scroll tables below by horizontal.
### Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = simavr
```

More options:

- **Debugging options**

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>Nuclei</td>
<td>On-board</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HummingBird Evaluation Kit</td>
<td>Nuclei</td>
<td>On-board</td>
<td>HUMMING-BIRD</td>
<td>5MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**simavr**

simavr is a lean, mean and hackable AVR simulator. Official reference can be found [here](#).

---

1.14. Debugging
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMega128/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega128</td>
<td>16MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ATMega1280</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1280</td>
<td>16MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ATMega1281</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1281</td>
<td>16MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ATMega1284</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1284</td>
<td>16MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ATMega1284P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>16MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ATMega16</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega16</td>
<td>16MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>ATMega164P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega164P</td>
<td>16MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>ATMega168/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega168</td>
<td>16MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>ATMega168/P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega168P</td>
<td>16MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>ATMega2560</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega2560</td>
<td>16MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ATMega324A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega324A</td>
<td>16MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ATMega324P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega324P</td>
<td>16MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ATMega324PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega324PA</td>
<td>16MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ATMega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328</td>
<td>16MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ATMega328/P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ATMega48/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega48</td>
<td>16MHz</td>
<td>4KB</td>
</tr>
<tr>
<td>ATMega48/P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega48P</td>
<td>16MHz</td>
<td>4KB</td>
</tr>
<tr>
<td>ATMega644/P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega644P</td>
<td>16MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ATMega8/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega8</td>
<td>16MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>ATMega88/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega88</td>
<td>16MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>ATMega88/P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega88P</td>
<td>16MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>ATtiny13</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTiny13</td>
<td>9MHz</td>
<td>1KB</td>
</tr>
<tr>
<td>ATtiny13A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTiny13A</td>
<td>9MHz</td>
<td>1KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit Micro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Feather 328P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>Adafruit Feather 32u4</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTiny85</td>
<td>8MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>12MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (USB)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>12MHz</td>
<td>28KB</td>
</tr>
</tbody>
</table>

Continued on
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (USB)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Adafruit Trinket 3V/8MHz</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>Adafruit Trinket 5V/16MHz</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Alorium Hin</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Anarduino MiniWireless</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Arduboy</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduboy DevKit</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino BT ATmega168</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega168</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega328</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Explora</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Industrial 101</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Leonardo ETH</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino LilyPad USB</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>124KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega168</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega8</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega168</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Yun Mini</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>BQ ZUM BT-328</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>BitWizard Rasplduino</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Controllino Maxi</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>Amel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
</tbody>
</table>

Continued on
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digispark USB</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>5.87KB</td>
</tr>
<tr>
<td>Engduino 3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>EnvironDIY Mayfly</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>FYSETC F6 V1.3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
</tr>
<tr>
<td>Generic ATtiny2313</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY25</td>
<td>8MHz</td>
<td>2KB</td>
</tr>
<tr>
<td>Generic ATtiny4313</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY4313</td>
<td>8MHz</td>
<td>4KB</td>
</tr>
<tr>
<td>Generic ATtiny44</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY44</td>
<td>8MHz</td>
<td>4KB</td>
</tr>
<tr>
<td>Generic ATtiny45</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY45</td>
<td>8MHz</td>
<td>4KB</td>
</tr>
<tr>
<td>Generic ATtiny84</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY84</td>
<td>8MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>Generic ATtiny85</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>LightBlue Bean</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>LightUp</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Linino One</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>LinkIt Smart 7688 Duo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>LoRa32u4II (868-915MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>LowPowerLab MightyHat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>LowPowerLab MoteinoMEGA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega168PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega168PA@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168P</td>
<td>8MHz</td>
<td>15.50KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>Microduino Core USB (Atmega32U4@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega1284P@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega1284P@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@8M,3,3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
</tr>
<tr>
<td>OpenEnergyMonitor emonPi</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Original Prusa i3 MK3 Multi Material 2.0 Upgrade</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>PanStamp AVR</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>Pololu A-Star 32U4</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Prusa RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
</tr>
<tr>
<td>Quirkbot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>RedBearLab Blend</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/16MHz (overclock)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>RepRap RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
</tr>
<tr>
<td>SODAQ GaLoRa</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>SODAQ Mbili</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>SODAQ Moja</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
</tr>
<tr>
<td>SODAQ Ndogo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>SODAQ Tatu</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>Sanguino Atmega1284p (16MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>Sanguino Atmega1284p (8MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA644</td>
<td>16MHz</td>
<td>63KB</td>
</tr>
</tbody>
</table>
Table 62 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega644</td>
<td>8MHz</td>
<td>63KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega644P</td>
<td>16MHz</td>
<td>63KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega644P</td>
<td>8MHz</td>
<td>63KB</td>
</tr>
<tr>
<td>Seeeduino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>SparkFun ATmega128RFA1 Dev Board</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega128RFA1</td>
<td>16MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun Fio V3 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>SparkFun Makey Makey</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega2560</td>
<td>8MHz</td>
<td>252KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro Mini 3.3V</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>252KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>SparkFun Qduino Mini</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>SpellFoundry Sleepy Pi 2</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Talk2 Whisper Node</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>The Things Uno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>TinyCircuits TinyDuino Processor Board</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>USBasp stick</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega8</td>
<td>12MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V2</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>16MHz</td>
<td>120.00KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>16MHz</td>
<td>127KB</td>
</tr>
<tr>
<td>ftDuino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>nicai-systems BOB3 coding bot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega8</td>
<td>8MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>nicai-systems NIBO 2 robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega128</td>
<td>16MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega16</td>
<td>15MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>20MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega16</td>
<td>15MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega1284P</td>
<td>20MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ubIQio Ardhat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
</tbody>
</table>

**Sipeed RV Debugger**
High-speed 3-IN-1 fast USB ARM/ESP32 JTAG, USB-to-RS232 virtual port and power supply 5VDC device. Official reference can be found here.

### Contents

- Configuration
- Drivers
- Wiring Connections
- Platforms
- Frameworks
- Boards

### Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini` (Project Configuration File)“:

```
[env:myenv]
platform = ...
board = ...
debbug_tool = sipeed-rv-debugger
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debbug_tool = sipeed-rv-debugger
upload_protocol = sipeed-rv-debugger
```

More options:

- Debugging options
- Upload options

### Drivers

**Windows**

- Step-by-step guide: Drivers, Zadig, Wiring
- Video tutorial

**Mac** macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

**Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.
Wiring Connections

<table>
<thead>
<tr>
<th>Sipeed RV Debugger Connector</th>
<th>Board Pin</th>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td></td>
<td>Digital ground</td>
</tr>
<tr>
<td>2</td>
<td>TDI</td>
<td></td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>6</td>
<td>TMS</td>
<td></td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>10</td>
<td>TCK</td>
<td></td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>8</td>
<td>TDO</td>
<td></td>
<td>Test Data Out pin</td>
</tr>
<tr>
<td>4</td>
<td>RST</td>
<td></td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigaDevice GD32V</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>GigaDevice GD32V SDK</td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

**ST-LINK**

The ST-LINK is an in-circuit debugger and programmer for the STM8 and STM32 microcontroller families. The single wire interface module (SWIM) and JTAG/serial wire debugging (SWD) interfaces are used to communicate with any STM8 or STM32 microcontroller located on an application board. Official reference can be found here.

**Contents**

- **Configuration**
- **Drivers**
- **Wiring Connections**
  - JTAG Interface
– Serial Wire Mode Interface (SWD)

• Platforms
• Frameworks
• Boards

### Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = stlink
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = stlink
upload_protocol = stlink
```

More options:

• Debugging options
• Upload options

### Drivers

**Windows** Please install official ST-LINK USB driver.

**Mac** Not required.

**Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.
Wiring Connections

### JTAG Interface

<table>
<thead>
<tr>
<th>ST-Link JTAG 20-Pin Connector</th>
<th>Board Pin</th>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>1</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td>4</td>
<td>Digital ground</td>
</tr>
<tr>
<td>5</td>
<td>TDI</td>
<td>5</td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>7</td>
<td>TMS</td>
<td>7</td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>9</td>
<td>TCK</td>
<td>9</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>13</td>
<td>TDO</td>
<td>13</td>
<td>Test Data Out pin</td>
</tr>
<tr>
<td>15</td>
<td>RESET</td>
<td>15</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>

### Serial Wire Mode Interface (SWD)

<table>
<thead>
<tr>
<th>ST-Link SWD 20-Pin Connector</th>
<th>Board Pin</th>
<th>SWD Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>1</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td>4</td>
<td>Digital ground</td>
</tr>
<tr>
<td>7</td>
<td>SWdio</td>
<td>7</td>
<td>Data I/O</td>
</tr>
<tr>
<td>9</td>
<td>SWclock</td>
<td>9</td>
<td>Clock</td>
</tr>
<tr>
<td>15</td>
<td>RESET</td>
<td>15</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>
Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aceinna IMU</td>
<td>Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.</td>
</tr>
<tr>
<td>Atmel SAM</td>
<td>Atmel SMART offers Flash-based ARM products based on the ARM Cortex-M0+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.</td>
</tr>
<tr>
<td>Nordic nRF51</td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazell.</td>
</tr>
<tr>
<td>Nordic nRF52</td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td>ST STM32</td>
<td>The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.</td>
</tr>
<tr>
<td>ST STM8</td>
<td>The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.</td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>CMSIS</td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.</td>
</tr>
<tr>
<td>Mbed</td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>ST Standard Peripheral Library</td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind.</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others.</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1N</td>
</tr>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1N</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
<td>512K</td>
</tr>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512K</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2N</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1N</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.9G</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>----------</td>
<td>--------</td>
<td>--------------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>96Boards Nitrogen</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>51</td>
</tr>
<tr>
<td>Aceinna Low Cost RTK</td>
<td>Aceinna IMU</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1N</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1N</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1N</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32L431CB</td>
<td>80MHz</td>
<td>12</td>
</tr>
<tr>
<td>Aceinna OpenRTK330L</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F469I</td>
<td>180MHz</td>
<td>1N</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>51</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>79</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1N</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>79</td>
</tr>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>12</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>51</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>51</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1N</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.5</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>51</td>
</tr>
<tr>
<td>BBC micro:bit v2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>51</td>
</tr>
<tr>
<td>BL652 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>51</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1N</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>51</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>51</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>51</td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1N</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>12</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>25</td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>25</td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>51</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>51</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>12</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>51</td>
</tr>
<tr>
<td>BlueDK</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>25</td>
</tr>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>19</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>79</td>
</tr>
<tr>
<td>Core board F401RCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RCT6</td>
<td>84MHz</td>
<td>25</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>19</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>51</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>51</td>
</tr>
<tr>
<td>Econode-L082CZ</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>19</td>
</tr>
<tr>
<td>ElectronutLabs Blip</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1N</td>
</tr>
<tr>
<td>ElectronutLabs Papyr</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>STM32H750IWK6</td>
<td>400MHz</td>
<td>51</td>
</tr>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>51</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>51</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>51</td>
</tr>
</tbody>
</table>

1.14. Debugging
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>FK407/M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>FYSETC S6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>512KB</td>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Holyiot TJ-16019</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>IsyBitsy nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>512KB</td>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>512KB</td>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>MakenRdky nRF52832-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>maple Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>512KB</td>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>512KB</td>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F0405RGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>512KB</td>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>512KB</td>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
<td>OLMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>512KB</td>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>P-Nucleo WB55SRG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32WB55SRG</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>PYBStick26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>512KB</td>
<td>PYBStick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>PYBStick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>Particle Xenon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Piconomix PX-HER0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>512KB</td>
<td>PntrnBoard V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>512KB</td>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>512KB</td>
<td>Raytac MDBT50Q-RX Dongle</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Raytac MDBT50Q-RX Dongle</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
<td>Raytac MDBT50Q-RX Dongle</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RE76</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RG76</td>
<td>80MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Ruvi Tag</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VC76</td>
<td>84MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VE76</td>
<td>100MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413HZ76</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NI76</td>
<td>180MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NI76</td>
<td>216MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RC76</td>
<td>32MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VG76</td>
<td>80MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AG76</td>
<td>80MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST B-L475E-10T01A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VG76</td>
<td>80MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RE76</td>
<td>84MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Table 63 – continued from previous page

1.14. Debugging
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32H743ZI</td>
<td>400MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo H743ZI-Q</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32H743ZI</td>
<td>480MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L011K4T6</td>
<td>32MHz</td>
<td>16MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L031K6T6</td>
<td>32MHz</td>
<td>32MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L035R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L053R8T6</td>
<td>32MHz</td>
<td>64MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L073RZ</td>
<td>32MHz</td>
<td>19MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L152RE</td>
<td>32MHz</td>
<td>51MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L412KB</td>
<td>80MHz</td>
<td>12MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L432KC</td>
<td>80MHz</td>
<td>25MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L433RC</td>
<td>80MHz</td>
<td>25MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L452RE</td>
<td>80MHz</td>
<td>25MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L476RG</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L486RG</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L496ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L496ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L4R5ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F030R8T6</td>
<td>48MHz</td>
<td>64MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F051R8T6</td>
<td>48MHz</td>
<td>64MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F303VC</td>
<td>72MHz</td>
<td>12MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L073ZT6</td>
<td>32MHz</td>
<td>19MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L152RBT6</td>
<td>32MHz</td>
<td>12MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F100RBT6</td>
<td>24MHz</td>
<td>12MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32S-DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32S103S</td>
<td>16MHz</td>
<td>32MB</td>
<td></td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F401CCU6</td>
<td>84MHz</td>
<td>25MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32J0C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F107VCT6</td>
<td>72MHz</td>
<td>25MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32J3C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F373VCT6</td>
<td>72MHz</td>
<td>25MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32M02EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F072VB</td>
<td>48MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103C4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103C4</td>
<td>72MHz</td>
<td>16MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103C6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103C6</td>
<td>72MHz</td>
<td>32MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103C8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103C8T6</td>
<td>72MHz</td>
<td>64MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103CB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103CBT6</td>
<td>72MHz</td>
<td>12MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103CR (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103CR</td>
<td>72MHz</td>
<td>16MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103R6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103R6</td>
<td>72MHz</td>
<td>32MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103R8 (20k RAM. 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103R8T6</td>
<td>72MHz</td>
<td>64MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RB</td>
<td>72MHz</td>
<td>12MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RC (48k RAM. 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RC</td>
<td>72MHz</td>
<td>25MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RD (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RD</td>
<td>72MHz</td>
<td>38MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RE (64k RAM. 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RE</td>
<td>72MHz</td>
<td>51MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RF (96k RAM. 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RF</td>
<td>72MHz</td>
<td>76MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RG (96k RAM. 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RT4 (6k RAM. 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RT4</td>
<td>72MHz</td>
<td>16MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RT6 (10k RAM. 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RT6</td>
<td>72MHz</td>
<td>32MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RT8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RT8T6</td>
<td>72MHz</td>
<td>64MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RTB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RTB6</td>
<td>72MHz</td>
<td>12MB</td>
<td></td>
</tr>
<tr>
<td>ST STM32F103RV8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RV8</td>
<td>72MHz</td>
<td>64MB</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F103VB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F103VC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256k</td>
</tr>
<tr>
<td>STM32F103VD</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103VF</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VFT6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VGT6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103ZC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103ZD</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103ZF</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZF6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F103ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZG6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F303CB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F401CB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CB</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CC</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F401CD</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CD</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CE</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F401RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RB</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F401RC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RC</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F401RD</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RD</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F401RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RE</td>
<td>84MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F405RG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>256k</td>
</tr>
<tr>
<td>STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F410C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410C8</td>
<td>100MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F410CB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F410R8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410R8</td>
<td>100MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F410RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128k</td>
</tr>
<tr>
<td>STM32F411CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256k</td>
</tr>
<tr>
<td>STM32F411CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>256k</td>
</tr>
<tr>
<td>STM32F411RC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256k</td>
</tr>
<tr>
<td>STM32F412CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F412CG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F412RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F412RG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F413CG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F413CH</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F413RG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F413RH</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RH</td>
<td>100MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F415RG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F417VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F417VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F423CH</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F423RH</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F446RC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F446RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512k</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1024k</td>
</tr>
<tr>
<td>STM32F750-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64k</td>
</tr>
<tr>
<td>STM32H747-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747XIH6</td>
<td>400MHz</td>
<td>256k</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>-------</td>
<td>-----</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>8KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439V1</td>
<td>180MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>12MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ThunderPack v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>reel_board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>512KB</td>
</tr>
</tbody>
</table>

**TI-ICDI**

Tiva™ C Series evaluation and reference design kits provide an integrated In-Circuit Debug Interface (ICDI) which allows programming and debugging of the onboard C Series microcontroller. Official reference can be found [here](#).
Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = ti-icdi
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = ti-icdi
upload_protocol = ti-icdi
```

More options:

- Debugging options
- Upload options

Drivers

Windows  Please “USB Driver Installation” guide for your board.

Mac  Not required.

Linux  Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI</td>
<td>Texas Instruments TM4C12x MCUs offer the industry's most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>libopenCM3</td>
<td>The libOpenCM3 framework aims to create a free and open-source firmware library for various ARM Cortex-M0(+)/M3/M4 microcontrollers, including ST STM32, Ti Tiva and Stellaris, NXP LPC, Atmel SAM3, Energy Micro EFM32 and others</td>
</tr>
</tbody>
</table>
Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>TI</td>
<td>TIVA</td>
<td>On-board</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>TI</td>
<td>TIVA</td>
<td>On-board</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>TI</td>
<td>TIVA</td>
<td>On-board</td>
<td>LPTM4C1294NCPI</td>
<td>120MHz</td>
<td>1MB</td>
</tr>
</tbody>
</table>

**TIAO USB Multi-Protocol Adapter (TUMPA)**

The TIAO USB Multi Protocol Adapter (TUMPA) is a multi-functional USB communication adapter for hobbyists or engineers. The adapter is based on FDTI's flagship communication chip FT2232H, a USB 2.0 Hi-Speed (480Mb/s) to UART/FIFO IC. It has two multi-protocol synchronous serial engines (MPSSEs) which allow for communication using JTAG, I2C and SPI on two channels simultaneously. Official reference can be found [here](#).

**Contents**

- Configuration
- Drivers
- Wiring Connections
  - JTAG Interface
  - Serial Wire Mode Interface (SWD)
- Platforms
- Frameworks
- Boards
Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debuge_tool = tumpa
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debuge_tool = tumpa
upload_protocol = tumpa
```

More options:
- `Debugging options`
- `Upload options`

Drivers

- **Windows**
  - Step-by-step guide: Drivers, Zadig, Wiring
  - Video tutorial

- **Mac** macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

- **Linux** Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.
Wiring Connections

### JTAG Interface

<table>
<thead>
<tr>
<th>Board JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 VCC</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4 GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>5 TDI</td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>7 TMS</td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>9 TCK</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>13 TDO</td>
<td>Test Data Out pin</td>
</tr>
<tr>
<td>15 RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)</td>
</tr>
</tbody>
</table>

### Serial Wire Mode Interface (SWD)

<table>
<thead>
<tr>
<th>Board SWD Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 VCC</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4 GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>7 SWDIO</td>
<td>Data I/O</td>
</tr>
<tr>
<td>9 SWCLK</td>
<td>Clock</td>
</tr>
<tr>
<td>15 RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
</tbody>
</table>
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td>ESP-IDF</td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

## Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 64 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOIT ESP32 DEVKit V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESP32yn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ESPressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>IoTaaP Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUNIO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-T Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-T Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
The UM232H is a USB-to-serial/FIFO development module in the FTDI product range which utilizes the FT232H USB Hi-Speed (480Mb/s) single-port bridge chip to handle the USB signaling and protocols. Official reference can be found here.

**Contents**

- Configuration
- Drivers
- Wiring Connections
- Platforms
- Frameworks
- Boards

**Configuration**

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
download_tool = um232h
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
download_tool = um232h
upload_protocol = um232h
```

More options:
• Debugging options
• Upload options

Drivers

Windows

• Step-by-step guide: Drivers, Zadig, Wiring
• Video tutorial

Mac macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) AN134: FTDI Drivers Installation guide for MAC OS X

Linux Please install “udev” rules 99-platformio-udev.rules. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Wiring Connections

Please read 4. UM232H Pin Out and Signal Descriptions section for details.

<table>
<thead>
<tr>
<th>UM232H Pin</th>
<th>Board JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>AD0</td>
<td>TCK</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>AD1</td>
<td>TDI</td>
<td>Test Data In</td>
</tr>
<tr>
<td>AD2</td>
<td>TDO</td>
<td>Test Data Out</td>
</tr>
<tr>
<td>AD3</td>
<td>TMS</td>
<td>Test Mode State</td>
</tr>
</tbody>
</table>

You will also need to connect VIO to V3V and USB to 5V0 of UM232H to power the FTDI chip and board. See UM232H Datasheet

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GigaDevice</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td>GD32V</td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>GigaDevice</strong></td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
</tbody>
</table>

Boards

**Note:** For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Verilator

Verilator is a free and open-source software tool which converts Verilog to a cycle-accurate behavioral model in C++ or SystemC. Official reference can be found here.
Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
d debug_tool = verilator
```

More options:

- Debugging options

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>WD-Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards

Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
</tbody>
</table>
Whisper

Whisper is a RISCV instruction set simulator (ISS) developed for the verification of the Swerv micro-controller. It allows the user to run RISCV code without RISCV hardware. It has an interactive mode where the user can single step the target RISCV code and inspect/modify the RISCV registers or the simulated system memory. It can also run in lock step with a Verilog simulator serving as a “golden model” against which an implementation is checked after each instruction of a test program. Official reference can be found here.

Contents

- Configuration
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```ini
[env:myenv]
platform = ...
broadcast = ...
debug_tool = whisper
```

More options:

- Debugging options

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPS Alliance</td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeRTOS</td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td>WD Firmware</td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td>Zephyr RTOS</td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>

Boards
Note: For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
<td></td>
</tr>
</tbody>
</table>

Custom

**Configuration**  `debug_tool = custom`

**PlatformIO Debugging Solution** can be configured using "platformio.ini" (Project Configuration File):

**Examples**

- J-Link and ST Nucleo
- J-Link as debugger and uploader
- ST-Util and ST-Link
- OpenOCD and ST-Link
- pyOCD and CMSIS-DAP

**J-Link and ST Nucleo**

Segger J-Link probe and ST Nucleo F446RE board in pair with J-Link GDB Server:

- Install J-Link GDB Server
- Convert ST-LINK On-Board Into a J-Link

**Note:** You can use configuration below in pair with other boards, not only with ST Nucleo F446RE. In this case, please replace `STM32F446RE` with your own device name in `debug_server` option.

See full list with J-Link Supported Devices.

```ini
[env:debug_jlink]
platform = ststm32
framework = mbed
board = nucleo_f446re

debug_tool = custom
debug_port = :2331
debug_server = /full/path/to/JLinkGDBServerCL -singlerun -if SWD
```

(continues on next page)
J-Link as debugger and uploader

Segger J-Link probe as debugger and uploader for a custom board. If you plan to use with other board, please change device MK20DX256xxx7 to a valid identifier. See supported J-Link devices at J-LINK.

- Install J-Link GDB Server

```ini
[env:jlink_debug_and_upload]
platform = teensy
framework = arduino
board = teensy31
extra_scripts = extra_script.py
upload_protocol = custom
debug_tool = jlink
debug_server = /full/path/to/JLinkGDBServerCL
-singlerun
-if
-swd
-select
usb
-port
2331
-device
MK20DX256xxx7
```

extra_script.py

Place this file on the same level as “platformio.ini” (Project Configuration File).
from os import makedirs
from os.path import isdir, join
Import('env')

def _jlink_cmd_script(env, source):
    build_dir = env.subst("$BUILD_DIR")
    if not isdir(build_dir):
        makedirs(build_dir)
    script_path = join(build_dir, "upload.jlink")
    commands = ["h", "loadbin %s,0x0" % source, "r", "q"]
    with open(script_path, "w") as fp:
        fp.write("\n".join(commands))
    return script_path

env.Replace(__jlink_cmd_script=_jlink_cmd_script,
            UPLOADER="/full/path/to/JLink",
            UPLOADERFLAGS=[
                "-device", "MK20DX256xxx7",
                "-speed", "4000",
                "-if", "swd",
                "-autoconnect", "1"
            ],
            UPLOADCMD="$UPLOADER $UPLOADERFLAGS -CommanderScript ${__jlink_cmd_script(__env__, SOURCE)}")

ST-Util and ST-Link

On-board ST-Link V2/V2-1 in pair with ST-Util GDB Server:

[env:debug]
platform = ststm32
framework = mbed
board = ...
debug_tool = custom
debug_port = :4242
debug_server = $PLATFORMIO_CORE_DIR/packages/tool-stlink/bin/st-util

OpenOCD and ST-Link

On-board ST-Link V2/V2-1 in pair with OpenOCD GDB Server:

[env:debug]
platform = ststm32
framework = mbed
board = ...
debug_tool = custom
debug_port = $PLATFORMIO_CORE_DIR/packages/tool-openocd/bin/openocd
debug_server =
  $PLATFORMIO_CORE_DIR/packages/tool-openocd/scripts/board/st_nucleo_f4.cfg
pyOCD and CMSIS-DAP

Using pyOCD for CMSIS-DAP based boards

Firstly, please install pyOCD and check that `pyocd-gdbserver --version` command works.

```python
[env:debug]
platform = ...
board = ...
framework = mbed
debug_tool = custom
debug_server = pyocd-gdbserver
```
1.14.4 CLI Guide

1.14.5 Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Aceinna IMU</strong></td>
<td>Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.</td>
</tr>
<tr>
<td>Atmel AVR 8-bit</td>
<td>Atmel AVR 8-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed time to market and easily adapt to new ones - they are based on the industry's most code-efficient architecture for C and assembly programming.</td>
</tr>
<tr>
<td>Atmel SAM</td>
<td>Atmel SMART offers Flash-based ARM products based on the ARM Cortex-MO+, Cortex-M3 and Cortex-M4 architectures, ranging from 8KB to 2MB of Flash including a rich peripheral and feature mix.</td>
</tr>
<tr>
<td><strong>CHIPS Alliance</strong></td>
<td>The CHIPS Alliance develops high-quality, open source hardware designs relevant to silicon devices and FPGAs.</td>
</tr>
<tr>
<td>Espressif 32</td>
<td>Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.</td>
</tr>
<tr>
<td><strong>Freescale Kinetics</strong></td>
<td>Freescale Kinetics Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetics MCUs offer exceptional low-power performance, scalability and feature integration.</td>
</tr>
<tr>
<td>GigaDevice GD32V</td>
<td>The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.</td>
</tr>
<tr>
<td>Infineon XMC</td>
<td>Infineon has designed the XMC microcontrollers for real-time critical applications with an industry-standard core. The XMC microcontrollers can be integrated with the Arduino platform.</td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC.</td>
</tr>
<tr>
<td>Maxim 32</td>
<td>Maxim’s microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim’s processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.</td>
</tr>
<tr>
<td>Nordic nRF51</td>
<td>The Nordic nRF51 Series is a family of highly flexible, multi-protocol, system-on-chip (SoC) devices for ultra-low power wireless applications. nRF51 Series devices support a range of protocol stacks including Bluetooth Smart (previously called Bluetooth low energy), ANT and proprietary 2.4GHz protocols such as Gazelle.</td>
</tr>
<tr>
<td>Nordic nRF52</td>
<td>The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.</td>
</tr>
<tr>
<td>NXP i.MX RT</td>
<td>The i.MX RT series of crossover processors features the Arm Cortex-M core, real-time functionality and MCU usability at a cost-effective price.</td>
</tr>
<tr>
<td><strong>NXP LPC</strong></td>
<td>The NXP LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors. The LPC chips are grouped into related series that are based around the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0. Internally, each microcontroller consists of the processor core, static RAM memory, flash memory, debugging interface, and various peripherals.</td>
</tr>
<tr>
<td><strong>RISC-V GAP</strong></td>
<td>GreenWaves GAP8 IoT application processor enables the cost-effective development, deployment and autonomous operation of intelligent sensing devices that capture, analyze, classify and act on the fusion of rich data sources such as images, sounds or vibrations.</td>
</tr>
<tr>
<td>Shakti</td>
<td>Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, high-performance grade processors, but also associated components like interconnect fabrics, verification cores, storage controllers, peripheral IPs and SOC tools.</td>
</tr>
<tr>
<td>SiFive</td>
<td>SiFive brings the power of open source and software automation to the semiconductor industry, making it possible to develop new hardware faster and more affordably than ever before.</td>
</tr>
<tr>
<td><strong>Silicon Labs EFM32</strong></td>
<td>Silicon Labs EFM32 Gecko 32-bit microcontroller (MCU) family includes devices that offer flash memory, ultra-low power consumption and high performance.</td>
</tr>
</tbody>
</table>

1.14. Debugging 2543
### 1.14.6 Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arduino</strong></td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences</td>
</tr>
<tr>
<td><strong>CMSIS</strong></td>
<td>The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices</td>
</tr>
<tr>
<td><strong>Espresif IoT Development Framework</strong></td>
<td>ESP-IDF is the official development framework for the ESP32 and ESP32-S Series SoCs.</td>
</tr>
<tr>
<td><strong>FreeRTOS</strong></td>
<td>FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to 40 microcontroller platforms</td>
</tr>
<tr>
<td><strong>Freedom E SDK</strong></td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td><strong>GigaDevice GD32V SDK</strong></td>
<td>GigaDevice GD32VF103 Firmware Library (SDK) is a firmware function package, including programs, data structures and macro definitions, all the performance features of peripherals of GD32VF103 devices are involved in the package</td>
</tr>
<tr>
<td><strong>Kendryte FreeRTOS SDK</strong></td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td><strong>Kendryte Standalone SDK</strong></td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td><strong>Mbed</strong></td>
<td>Arm Mbed OS is an open source embedded operating system designed specifically for the ‘things’ in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS and drivers for sensors and I/O devices</td>
</tr>
<tr>
<td><strong>Nuclei SDK</strong></td>
<td>Open Source Software Development Kit for the Nuclei N/NX processors</td>
</tr>
<tr>
<td><strong>PULP OS</strong></td>
<td>PULP is a silicon-proven Parallel Ultra Low Power platform targeting high energy efficiencies. The platform is organized in clusters of RISC-V cores that share a tightly-coupled data memory</td>
</tr>
<tr>
<td><strong>STM32Cube</strong></td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency</td>
</tr>
<tr>
<td><strong>Shakti SDK</strong></td>
<td>A software development kit for developing applications on Shakti class of processors</td>
</tr>
<tr>
<td><strong>Simba</strong></td>
<td>Simba is an RTOS and build framework with aims to make embedded programming easy and portable</td>
</tr>
<tr>
<td><strong>Standard Peripheral Library</strong></td>
<td>The ST Standard Peripheral Library provides a set of functions for handling the peripherals on the STM32 family of microcontrollers.</td>
</tr>
<tr>
<td><strong>WD Firmware</strong></td>
<td>The WD Firmware package contains firmware applications and Processor Support Package (PSP) for various cores, alongside demos which support all features</td>
</tr>
<tr>
<td><strong>Zephyr RTOS</strong></td>
<td>The Zephyr Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with safety and security in mind</td>
</tr>
</tbody>
</table>
1.14.7 Boards

Note: For more detailed board information please scroll tables below by horizontal.

### 1BitSquared

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Bitsy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### 96Boards

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards Argonkey (STEVAL-MKI187V1)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>96Boards Neonkey</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>96Boards Nitrogen</td>
<td>Nordic NRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### AI Thinker

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI Thinker ESP32-CAM</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### AZ-Delivery

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ-Delivery ESP-32 Dev Kit C V4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
</tbody>
</table>

### Aceinna

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aceinna Low Cost RTK</td>
<td>Aceinna IMU</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32L431CB</td>
<td>80MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Aceinna OpenIMU 330ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F469IG</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>Aceinna OpenRTK330L</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F469IG</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------</td>
<td>-------------</td>
<td>--------------</td>
<td>-----------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>Adafruit BLM Badge</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit Micro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5KB</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit CLUE nRF52840</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Cricket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Express 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Adafruit Feather 32P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Feather 32u4</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5KB</td>
</tr>
<tr>
<td>Adafruit Feather Bluefruit Sense</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 CAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAME51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAME51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Feather STM32F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Hallowing M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.5KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.5KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit MONSTER M4SK</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Matrix Portal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Metro M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M4 AirLift Lite</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (USB)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (USB)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyPortal M4 Titano</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit QT Py M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Trellis M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Trinket 3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit Trinket 5V/16MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 65 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit pyBadge AirLift M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1008KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit pyBadge M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Circuit Playground Bluefruit</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>ItsyBitsy nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Metro nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

AfroFlight

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AfroFlight Rev5 (8MHz)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Airbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wraith V1 ESC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051K6</td>
<td>48MHz</td>
<td>32KB</td>
<td>7.75KB</td>
</tr>
</tbody>
</table>

Aiyarafun

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

Alorium Technology

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alorium Hinj</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Anarduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anarduino MiniWireless</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Arduboy

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduboy</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduboy DevKit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>
## Arduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino BT ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Dicemila ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Esplora</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Industrial 101</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Leonardo ETH</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino LilyPad USB</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino M0 Pro (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKR WAN 1310</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>124KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega8</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
</tr>
<tr>
<td>Arduino Nano 33 BLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>960KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega168</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 66 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Yun Mini</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td></td>
</tr>
</tbody>
</table>

#### Armed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

#### Armstrap

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

#### Atmel

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMC21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMC21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Generic ATtiny2313</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY25</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny4313</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY4313</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny44</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY44</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny45</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY45</td>
<td>8MHz</td>
<td>4KB</td>
<td>256B</td>
</tr>
<tr>
<td>Generic ATtiny84</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY84</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Generic ATtiny85</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>USBasp stick</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>12MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
</tbody>
</table>

#### Avnet Silica

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### BBC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>BBC micro:bit V2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52833</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### BQ

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BQ ZUM BT-328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### BSFrance

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoRa32u4II (868-915MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### BitWizard

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitWizard Raspduino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### BluzDK

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BluzDK</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### CQ Publishing

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

### Calliope

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calliope mini</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>
## Controllino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllino Maxi</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATmega2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATmega2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATmega2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATmega328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

## DFRobot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## DOIT

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## DSTIKE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-duino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Delta

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Delta DFCM-NNN40</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>nRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>nRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## Digilent

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVfpga: Digilent Nexys A7</td>
<td>CHIPS Alliance</td>
<td>On-board</td>
<td></td>
<td>320MHz</td>
<td>16MB</td>
<td>1.16MB</td>
</tr>
</tbody>
</table>

## Digistump

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digispark USB</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTiny85</td>
<td>16MHz</td>
<td>5.87KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>
### Diymore

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VG16</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Dongsen Technology

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### DycodeX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPectro32</td>
<td>Espressif32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### ESP32vn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32vn IoT Uno</td>
<td>Espressif32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Econode

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Econode-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Electronut Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### ElectronutLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElectronutLabs Blip</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ElectronutLabs Papyr</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### Electrosmith

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrosmith Daisy</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32H750IBK6</td>
<td>400MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### Elektor Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

### Embedded Artists

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

### Engduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engduino 3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### EnviroDIY

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnviroDIY Mayfly</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Espressif

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32 Pico Kit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### FYSETC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FYSETC F6 V1.3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>FYSETC 36</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

2554 Chapter 1. Contents
## PlatformIO Documentation, Release 5.0.5a1

### Fred

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Freescale

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK20DX128VHLH5</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK22F512VHLH12</td>
<td>120MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK66F2N0VMD</td>
<td>8180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK82F256VLL15</td>
<td>150MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL05Z32VFM4</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128VLSK4</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL27Z64VLSK4</td>
<td>48MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL43Z256VLSK4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL46Z256VLSK4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL82Z</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKL82Z128VLSK7</td>
<td>96MHz</td>
<td>128KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKW41Z512VHT4</td>
<td>48MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Generic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F103C4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103C6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CR (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CR</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103RT (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RT6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RTB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RD</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103RF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103RG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103T4 (6k RAM, 16k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T4</td>
<td>72MHz</td>
<td>16KB</td>
<td>6KB</td>
</tr>
<tr>
<td>STM32F103T6 (10k RAM, 32k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103V8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103V8</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103VG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZC (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZC6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZF (96k RAM, 768k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZF</td>
<td>72MHz</td>
<td>768KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F103ZG (96k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZG</td>
<td>72MHz</td>
<td>1MB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F3041CB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041CB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F3041CC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041CC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F3041CD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041CD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F3041CE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041CE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F3041RB (64k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041RB</td>
<td>84MHz</td>
<td>128KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F3041RC (64k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041RC</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F3041RD (96k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041RD</td>
<td>84MHz</td>
<td>384KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F3041RE (96k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F3041RE</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>STM32F405RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F4010C8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F4010C8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410CB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410R8 (32k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410R8</td>
<td>100MHz</td>
<td>64KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F410RB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F410RB</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F411CC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411CE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RC</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F411RE (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RE</td>
<td>100MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
Table 67 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F412CE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412CG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RE (256k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F412RG (256k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>STM32F413CG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413RG (320k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413RG</td>
<td>100MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F413CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F413CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F415RG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F415RG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F417VG (128k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F423CH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423CH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F423RH (320k RAM, 1536k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F423RH</td>
<td>100MHz</td>
<td>1.50MB</td>
<td>320KB</td>
</tr>
<tr>
<td>STM32F446RC (128k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RC</td>
<td>180MHz</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F446RE (128k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F405RGT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

GigaDevice

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V Evaluation Kit</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103VB76</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>Nuclei</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Gimasi

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

GreenWaves Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAPuino GAP8</td>
<td>RISC-V GAP</td>
<td>On-board</td>
<td>GAP8</td>
<td>250MHz</td>
<td>64MB</td>
<td>8MB</td>
</tr>
</tbody>
</table>

HY

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Heltec Automation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### Holyiot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holyiot YJ-16019</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### Hornbill

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Infineon

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMC1100 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>32KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>32KB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1400</td>
<td>48MHz</td>
<td>1.95MB</td>
<td>16KB</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4200</td>
<td>80MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4700</td>
<td>144MHz</td>
<td>2.00MB</td>
<td>1.95MB</td>
</tr>
</tbody>
</table>

### IoTaaP

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoTaaP Magnolia</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### JKSoft

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>JKSoft Wallbot BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Laird Connectivity

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL652 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BL654 Development Kit</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Laird Connectivity Pinnacle 100 DVK</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
LeafLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
</tbody>
</table>

LightUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightUp</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>2KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

Linino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linino One</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>2KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

LowPowerLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowPowerLab MightyHat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8Mhz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab MoteinoMEGA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

MH-ET Live

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

MXChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Azure IoT Development Kit (MX-</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZG</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Chip AZ3166)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.14. Debugging
### Makerdiary

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makerdiary nRF52832-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Makerdiary nRF52840-MDK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### Malyan

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>M300</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Maxim

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX32620FTHR</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32620FTHR</td>
<td>48MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Maxim ARM mbed Enabled Development Platform for MAX32600</td>
<td>Maxim 32</td>
<td>On-board</td>
<td>MAX32600</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Maxim Health Sensor Platform</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32620</td>
<td>96MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Maxim Wireless Sensor Node Demonstrator</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32610</td>
<td>24MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### MediaTek Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkIt Smart 7688 Duo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

---
### Microchip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega128/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega1280</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1281</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1281</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1284</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega1284P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega16</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA164P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>256KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega324A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA324A</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324P</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA324P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA324PA</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA48</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega48P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA48P</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega644P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>64KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega8/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88/A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA88</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88P/PA</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA88P</td>
<td>16MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATTiny13</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY13</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>ATTiny13A</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATTINY13A</td>
<td>9MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
</tbody>
</table>

1.14. Debugging

2561
## Microduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microduino Core (Atmega168PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega168PA@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA168P</td>
<td>8MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CB</td>
<td>62MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>Microduino Core USB (Atmega32U4@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega1284P@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega1284P@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@8M,3.3V)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

## Micromint

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
<td>204MHz</td>
<td>8MB</td>
<td>264KB</td>
</tr>
</tbody>
</table>

## Midatronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKR Sharky</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32WB55CG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192.00KB</td>
</tr>
</tbody>
</table>

## MikroElektronikka

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hexitwear</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
## MultiTech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

## NGX Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>
## NXP

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM mbed LPC11U24</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
<td>50MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11C24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11C24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U34</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U34</td>
<td>48MHz</td>
<td>40KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC11U37</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U37</td>
<td>48MHz</td>
<td>128KB</td>
<td>10KB</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>NXP LPCXpresso1549</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1549</td>
<td>72MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54114J256BD64</td>
<td>100MHz</td>
<td>256KB</td>
<td>192KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54608</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54608ET512</td>
<td>180MHz</td>
<td>512KB</td>
<td>200KB</td>
</tr>
<tr>
<td>NXP LPCXpresso55S16</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC55S16</td>
<td>150MHz</td>
<td>256KB</td>
<td>96KB</td>
</tr>
<tr>
<td>NXP LPCXpresso55S69</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC55S69</td>
<td>150MHz</td>
<td>640KB</td>
<td>320KB</td>
</tr>
<tr>
<td>NXP i.MX RT1010 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1011DAE5A</td>
<td>500MHz</td>
<td>64KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1015 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1015DAF5A</td>
<td>500MHz</td>
<td>96KB</td>
<td>128KB</td>
</tr>
<tr>
<td>NXP i.MX RT1020 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1021DAG5A</td>
<td>500MHz</td>
<td>64MB</td>
<td>256MB</td>
</tr>
<tr>
<td>NXP i.MX RT1050 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1052DVL6B</td>
<td>600MHz</td>
<td>64MB</td>
<td>512KB</td>
</tr>
<tr>
<td>NXP i.MX RT1060 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1062DVL6A</td>
<td>600MHz</td>
<td>64MB</td>
<td>1MB</td>
</tr>
<tr>
<td>NXP i.MX RT1064 Evaluation Kit</td>
<td>NXP i.MX RT</td>
<td>On-board</td>
<td>MIMXRT1064DVL6A</td>
<td>600MHz</td>
<td>4MB</td>
<td>1MB</td>
</tr>
<tr>
<td>NXP mbed LPC11U24</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP mbed LPC1768</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## Netduino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>
## NodeMCU

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## Nordic

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic Thingy:52 (nRF52-PCA20020)</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF51822-mKIT</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit(PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

## Nuclei

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V RVStar Kit</td>
<td>Nuclei</td>
<td>On-board</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>HummingBird Evaluation Kit</td>
<td>Nuclei</td>
<td>On-board</td>
<td>HUMMINGBIRD</td>
<td>5MHz</td>
<td>64KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## OLIMEX

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEX ESP32-DevKit-LiPo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## OSHChip

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
## PlatformIO Documentation, Release 5.0.5a1

### Olimex

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLIMEXINO-STM32</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-H103</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Olimex STM32-P405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32-E407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32-H407</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### OpenEnergyMonitor

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenEnergyMonitor emonPi</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### PHYTEC

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>reel_board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>reel_board_v2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### PYBStick

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYBSTICK26 Duino</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072RB</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>PYBStick 26 Pro</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F412RE</td>
<td>100MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>PYBStick Lite 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>PYBStick Standard 26</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### PanStamp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanStamp AVR</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Particle

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle Argon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Boron</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Particle Xenon</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>
### Piconomix

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piconomix PX-HERO</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L072RB</td>
<td>32MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

### Pololu Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pololu A-Star 32U4</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### PrntrBoard

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrntrBoard V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407RE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

### Prusa 3D

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Prusa i3 MK3 Multi Material 2.0 Upgrade</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Prusa RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### Punch Through

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightBlue Bean</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.5KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.5KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### Pycom Ltd.

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
</tbody>
</table>

### Quirkbot

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quirkbot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>
### RAK

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### RUMBA

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Raytac

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raytac MDBT50Q-RX Dongle</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
</tbody>
</table>

### RedBearLab

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/16MHz (overclock)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### RemRam

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</td>
</tr>
</tbody>
</table>

### RepRap

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepRap RAMBo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>
### ReprapWorld

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minitronics v2.0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### RobotDyn

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### RoboticsBrno

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### RushUp

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Ruuvi

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruuvi Tag</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### SEGGER

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEGGER IP Switch Board</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### SG-O

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG-O AirMon</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## SODAQ

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ ExplorER</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ GaLoRa</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Mbiti</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Moja</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SODAQ Ndogo</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ Tatu</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## ST

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>32F412GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>32F723EZHDiscovery</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723ZET6</td>
<td>216MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VGT6</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Black STM32F407ZG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>BlackPill F401CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>BlackPill F401CE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CEU6</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Core board F401RC16</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401RC16</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G071RBT6</td>
<td>128MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G431KB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431KBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G431RB</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G431RBT6</td>
<td>170MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>Nucleo G474RE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G474RET6</td>
<td>170MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32WB55RG</td>
<td>64MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>RHF76 052</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L051C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NIH6</td>
<td>216MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AG16</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST B-L475E-4I010A Discovery kit</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td></td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td></td>
</tr>
</tbody>
</table>

Continued
Table 68 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F070RB6T6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RB6T6</td>
<td>48MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RB6T6</td>
<td>72MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RE6T6</td>
<td>84MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F410RB6T6</td>
<td>100MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RE6T6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F446ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446ZET6</td>
<td>180MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F756ZG</td>
<td>216MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI-Q</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F743ZIT6</td>
<td>400MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RE6T6</td>
<td>80MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KBU6</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo L423KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L423KCU6</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RG16</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RG16</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L485ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L485ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F0308R8T6</td>
<td>48MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F0303VCT6</td>
<td>64MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST STM32G0316-DISCO</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32G031J6</td>
<td>64MHz</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073VZT6</td>
<td>32MHz</td>
<td>192KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RB6T6</td>
<td>32MHz</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued
### Table 68 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST STM32VL DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F100RB6</td>
<td>24MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>ST STM8S-DISCOVERY</td>
<td>ST STM8</td>
<td>On-board</td>
<td>STM8S105C6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td></td>
</tr>
<tr>
<td>STEVAL-FCU001V1 Flight controller unit evaluation board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F401CCU6</td>
<td>84MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>STM3210C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F107VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>STM32373C-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F373VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>STM32F072-EVAL</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F072VBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td></td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td></td>
</tr>
<tr>
<td>STM32H747I-DISCO</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H747XIH6</td>
<td>400MHz</td>
<td>2MB</td>
<td></td>
</tr>
<tr>
<td>SensorTile.box</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L4R9ZI</td>
<td>120MHz</td>
<td>2MB</td>
<td></td>
</tr>
</tbody>
</table>

### SainSmart

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

### Sanguino

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanguino ATmega1284p (16MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (16 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
</tbody>
</table>

### Seeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeeduino Femto M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Lite MG126</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Wio Terminal</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51P19A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeeduino XIAO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Seeeduino Zero</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
### SeeedStudio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Seeed Arch Pro</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Seeeduino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Wio Lite RISC-V</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### Semtech

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### SiFive

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive1</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### Sigma Delta Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### Silicognition

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## Silicon Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32GG990F1024</td>
<td>48MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32LG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32WG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32ZG222F32</td>
<td>24MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SLSTK3401A Pearl Gecko PG1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32PG1B200F256</td>
<td>40MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SLSTK3701A Giant Gecko S1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
<td>400MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
<td>400MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

## Sipeed

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD32VF103V-EVAL</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103VBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103CBT6</td>
<td>108MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Sipeed Longan Nano Lite</td>
<td>GigaDevice GD32V</td>
<td>External</td>
<td>GD32VF103C8T6</td>
<td>108MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
<tr>
<td>Sipeed MF1 MF1</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>

## Solder Splash Labs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DipCortex M3</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1347</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>50MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Name</td>
<td>Platform</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----------</td>
<td>----------------</td>
<td>----------------</td>
<td>-----------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>SparkFun 9DoF Razor IMU M0</td>
<td>Atmel</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun ATmega128RFA1 Dev Board</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA128RFA1</td>
<td>16MHz</td>
<td>16KB</td>
<td>124KB</td>
</tr>
<tr>
<td>SparkFun Digital Sandbox</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Fio V3 3.3V/8MHz</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun Makey Makey</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 3.3V/8MHz</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 5V/16MHz</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro Mini 3.3V</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 3.3V/8MHz</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 5V/16MHz</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Qduino Mini</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Qwiic Micro</td>
<td>Atmel</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun RED-V RedBoard</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RED-V Thing Plus</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun RedBoard Turbo</td>
<td>Atmel</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD21 Pro RF</td>
<td>Atmel</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SparkFun SAMD51 Thing Plus</td>
<td>Atmel</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>496KB</td>
<td>192KB</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>Atmel</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>
## SparkFun Electronics

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparkFun ESP32 Thing</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## SpellFoundry

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpellFoundry Sleepy Pi 2</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

## Switch Science

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch Science mbed HRM1017</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1114FN28</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Switch Science mbed TY51822r3</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
### TI

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI LaunchPad MSP-EXP430FR5739LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>15.37KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</td>
<td>TI TIVA</td>
<td>On-board</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</td>
<td>TI TIVA</td>
<td>On-board</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</td>
<td>TI TIVA</td>
<td>On-board</td>
<td>LPTM4C1294NCHD40MHz</td>
<td>1MB</td>
<td>256KB</td>
<td></td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430F5529</td>
<td>25MHz</td>
<td>47KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2311LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR2311</td>
<td>16MHz</td>
<td>3.75KB</td>
<td>1KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR2433LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR2433</td>
<td>8MHz</td>
<td>15KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR4133LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR4133</td>
<td>8MHz</td>
<td>15KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5969LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5994LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5994</td>
<td>16MHz</td>
<td>256KB</td>
<td>4KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>47KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2231</td>
<td>1MHz</td>
<td>2KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2452</td>
<td>16MHz</td>
<td>8KB</td>
<td>256B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2553LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430G2553</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
</tbody>
</table>

### TTGO

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V2</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>TTGO T1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### Taida Century

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

1.14. Debugging
## PlatformIO Documentation, Release 5.0.5a1

### TauLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparky V1 F303</td>
<td>STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
</tbody>
</table>

### Teensy

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>Teensy</td>
<td>External</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>Teensy</td>
<td>External</td>
<td>MK64FX512</td>
<td>120MHz</td>
<td>512KB</td>
<td>255.99KB</td>
</tr>
<tr>
<td>Teensy 3.6</td>
<td>Teensy</td>
<td>External</td>
<td>MK66FX1M0</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Teensy 4.0</td>
<td>Teensy</td>
<td>External</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>1.94MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy 4.1</td>
<td>Teensy</td>
<td>External</td>
<td>IMXRT1062</td>
<td>600MHz</td>
<td>7.75MB</td>
<td>512KB</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>Teensy</td>
<td>External</td>
<td>MKL26Z64</td>
<td>48MHz</td>
<td>62KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

### ThaiEasyElec

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### The Things Network

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Things Uno</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

### ThunderPack

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThunderPack v1.0</td>
<td>STM32</td>
<td>External</td>
<td>STM32L072KZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ThunderPack v1.1+</td>
<td>STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### Till Harbaum

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftDuino</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>
TinyCircuits

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyCircuits TinyDuino Processor</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>AT-MEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Tlera Corporation

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cicada-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Cricket-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Gnat-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Grasshopper-L082CZ</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L082CZY6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>

Unknown

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

VAE

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

VCCGND

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCCGND F103ZET6 Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

VNG

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNG VBLUNO51</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>VNG VBLUno52</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

VintLabs

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VintLabs ESP32 Devkit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
## WEMOS

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos D1 MINI ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

## WIZNet

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIZwiki-W7500</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500ECO</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500P</td>
<td>48MHz</td>
<td>128KB</td>
<td>48KB</td>
</tr>
</tbody>
</table>

## Waveshare

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Waveshare Open103Z</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

## WeAct

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeAct BlackPill V2.0 (BlackPill F411CE)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411CEU6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

## Wicked Device

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wicked Device WildFire V2</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>120.00KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V3</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

## Wisen

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk2 Whisper Node</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>
### Xilinx

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Arty A7-100: Artix-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td>Parasha on Artix-7 100T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Pinaka on Artix-7 35T Arty FPGA Evaluation Kit</td>
<td>Shakti</td>
<td>On-board</td>
<td>E-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128KB</td>
</tr>
<tr>
<td>Vajra on Arty A7-100: Artix-7 FPGA Development Board</td>
<td>Shakti</td>
<td>On-board</td>
<td>C-CLASS</td>
<td>50MHz</td>
<td>0B</td>
<td>128MB</td>
</tr>
</tbody>
</table>

### XinaBox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### decaWave

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>decaWave DWM1001 Module Development Board</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

### meteca

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Briki ABC (MBC-WB) - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki ABC (MBC-WB) - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Briki MBC-WB - ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>3.25MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Briki MBC-WB - Samd21</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

### ng-beacon

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

1.14. Debugging
### nica-systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>nica-systems BOB3 coding bot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA88</td>
<td>8MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nica-systems NIBO 2 robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>nica-systems NIBO burger robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nica-systems NIBO burger robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>nica-systems NIBObee robot</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA16</td>
<td>15MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nica-systems NIBObee robot with Tuning Kit</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### oddWires

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

### rhomb.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### sakura.io

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### sino:bit

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
u-blox

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

ubIQio

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubIQio Ardhat</td>
<td>Atmel AVR</td>
<td>On-board</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

y5 design

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>y5 LPC11U35 mbug</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
<td>64KB</td>
<td>10KB</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

1.15 Unit Testing

Contents
- Demo
- Tutorials and Examples
- Configuration
- Test Types
- Test Runner
- Test Transport
- Workflow
Unit Testing allows you segregating each part of the firmware/program and testing that the individual parts are working correctly. Using PlatformIO Unit Testing Solution you can execute the same tests on the local host machine (native), on the multiple local embedded devices/boards (connected to local host machine), or on both. When testing both, PlatformIO builds firmware on the host machine, uploads into a target device, starts tests, and collects the test results into test reports. The final information will be shown on the host side with informative output and statistic.

Using Remote Development you can start unit tests on the Remote Device from anywhere in the world or integrate with Continuous Integration systems.

1.15.1 Demo

This is a demo of Local & Embedded: Calculator, which demonstrates running embedded tests on physical hardware (Arduino Uno) and native tests on host machine (desktop).

Learn more about pio test command.
1.15.2 Tutorials and Examples

Tutorials

- Unit Testing of a “Blink” Project
- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing
- ThingForward: Start Embedded Testing with PlatformIO
- ThingForward: Embedded Testing with PlatformIO - Part 2
- ThingForward: Embedded Testing with PlatformIO – Part 3: Remoting
- ThingForward: Embedded Testing with PlatformIO – Part 4: Continuous Integration
- ThingForward, Webinar: Unit Testing for Embedded with PlatformIO and Qt Creator
- Xose Pérez: Automated unit testing in the metal

Project Examples

- Embedded: Wiring Blink
- Local & Embedded: Calculator
- PlatformIO Remote Unit Testing Example

For the other examples and source code please follow to PlatformIO Unit Testing Examples repository.

1.15.3 Configuration

PlatformIO Unit Testing Solution can be configured using "platformio.ini" (Project Configuration File)

1.15.4 Test Types

Desktop

PlatformIO Unit Testing Engine builds a test program for a host machine using Native development platform. This test could be run only with the desktop or Continuous Integration VM instance.

**Note:** PlatformIO does not install any toolchains automatically for Native and requires GCC toolchain to be installed on your host machine. Please open Terminal and check that the gcc command is installed.

Embedded

PlatformIO Unit Testing Engine builds a special firmware for a target device (board) and programs it. Then, it connects to this device using configured Serial test_port and communicates via test_transport. Finally, it runs tests on the embedded side, collects results, analyzes them, and provides a summary on a host machine side (desktop).

**Note:** Please note that the PlatformIO Unit Testing Engine uses the first available Serial/UART implementation (depending on a framework) as a communication interface between the PlatformIO Unit Testing Engine and
target device. If you use Serial in your project libraries, please wrap/hide Serial-based blocks with `#ifndef UNIT_TEST` macro.

Also, you can create custom `test_transport` and implement the base interface.

### 1.15.5 Test Runner

Test Runner allows you to process specific environments or ignore a test using “Glob patterns”. You can also ignore a test for specific environments using a `test_ignore` option from “platformio.ini” (Project Configuration File).

**Local**

Allows you to run a test on a host machine or on a target device (board), which is directly connected to the host machine. In this case, you need to use the `pio test` command.

**Remote**

Allows you to run test on a remote machine or remote target device (board) without having to depend on OS software, extra software, SSH, VPN or opening network ports. Remote Unit Testing works in pair with Remote Development. In this case, you need to use the special command `pio remote test`.

PlatformIO supports multiple Continuous Integration systems where you can run unit tests at each integration stage. See real PlatformIO Remote Unit Testing Example.

### 1.15.6 Test Transport

**PlatformIO Unit Testing Engine** uses different transports to communicate with a target device. By default, it uses Serial/UART transport provided by a framework. For example, when `framework = arduino`, the first available Serial will be used. When Native dev-platform is used a native transport will be activated automatically. See example below.

Default baudrate/speed is set to `test_speed`.

You can also define custom transport and implement its interface:

- `unittest_uart_begin();`
- `unittest_uart_putchar(char c);`
- `unittest_uart_flush();`
- `unittest_uart_end();`

**Examples**

1. Custom transport for Native platform
   
   - Set `test_transport = custom` in “platformio.ini” (Project Configuration File)

   ```
   [env:mycustomtransport]
   platform = native
   test_transport = custom
   ```

   - Create `unittest_transport.h` file in `project/test` directory and implement prototypes above
#ifndef UNITTEST_TRANSPORT_H
#define UNITTEST_TRANSPORT_H

#include <stdio.h>

void unittest_uart_begin() {
}

void unittest_uart_putchar(char c) {
    putchar(c);
}

void unittest_uart_flush() {
    fflush(stdout);
}

void unittest_uart_end() {
}

#endif

2. STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

1.15.7 Workflow

1. Create PlatformIO project using the `pio project init` command. For Desktop Unit Testing (on a host machine), you need to use `native`.

; PlatformIO Project Configuration File
;
; Platform: atmelavr
; Board: uno
[env:nodemcu]
platform = espressif8266
framework = arduino
board = nodemcuv2

; Desktop platforms (Win, Mac, Linux, Raspberry Pi, etc)
; See https://docs.platformio.org/page/platforms/native
(continues on next page)
[env:native]
platform = native

2. Create a test folder in a root of your project. See test_dir.
3. Write a test using API. Each test is a small independent program/firmware with its own main() or setup() / loop() functions. Test should start with UNITY_BEGIN() and finish with UNITY_END() calls.

**Warning:** If your board does not support software resetting via Serial.DTR/RTS, you should add at least 2 seconds delay before UNITY_BEGIN(). That time is needed to establish a Serial connection between a host machine and a target device.

delay(2000); // for Arduino framework
wait(2);     // for ARM mbed framework
UNITY_BEGIN();

4. Place a test in the test directory. If you have more than one test, split them into sub-folders. For example, test/test_1/*.c,cpp,h], test_N/*.c,cpp,h], etc. If there is no such directory in the test folder, then PlatformIO Unit Testing Engine will treat the source code of test folder as SINGLE test.
5. Run tests using the pio test command.

**Shared Code**

PlatformIO Unit Testing Engine does not build source code from src_dir folder by default. If you have a shared/common code between your “main” and “test” programs, you have 2 options:

1. **RECOMMENDED.** We recommend splitting the source code into multiple components and placing them into lib_dir (project’s private libraries and components). Library Dependency Finder (LDF) will find and include these libraries automatically in the build process. You can include any library/component header file in your test or program source code via #include <MyComponent.h>.

   See Local & Embedded: Calculator for an example, where we have a “calculator” component in lib_dir folder and include it in tests and the main program using #include <calculator.h>.

2. Manually instruct PlatformIO to build source code from src_dir folder using test_build_project_src option in “platformio.ini” (Project Configuration File):

   ![Code snippet]

   This is very useful if you unit test independent libraries where you can’t split source code.

   **Warning:** Please note that you will need to use #ifdef UNIT_TEST and #endif guard to hide non-test related source code. For example, own main() or setup() / loop() functions.

**1.15.8 API**

Summary of the Unity Test API:
• **Running Tests**
  - `RUN_TEST(func)`

• **Ignoring Tests**
  - `TEST_IGNORE()`
  - `TEST_IGNORE_MESSAGE(message)`

• **Aborting Tests**
  - `TEST_PROTECT()`
  - `TEST_ABORT()`

• **Basic Validity Tests**
  - `TEST_ASSERT_TRUE(condition)`
  - `TEST_ASSERT_FALSE(condition)`
  - `TEST_ASSERT(condition)`
  - `TEST_ASSERT_UNLESS(condition)`
  - `TEST_FAIL()`
  - `TEST_FAIL_MESSAGE(message)`

• **Numerical Assertions: Integers**
  - `TEST_ASSERT_EQUAL_INT(expected, actual)`
  - `TEST_ASSERT_EQUAL_INT8(expected, actual)`
  - `TEST_ASSERT_EQUAL_INT16(expected, actual)`
  - `TEST_ASSERT_EQUAL_INT32(expected, actual)`
  - `TEST_ASSERT_EQUAL_INT64(expected, actual)`
  - `TEST_ASSERT_EQUAL_UINT(expected, actual)`
  - `TEST_ASSERT_EQUAL_UINT8(expected, actual)`
  - `TEST_ASSERT_EQUAL_UINT16(expected, actual)`
  - `TEST_ASSERT_EQUAL_UINT32(expected, actual)`
  - `TEST_ASSERT_EQUAL_UINT64(expected, actual)`
  - `TEST_ASSERT_EQUAL_HEX(expected, actual)`
  - `TEST_ASSERT_EQUAL_HEX8(expected, actual)`
  - `TEST_ASSERT_EQUAL_HEX16(expected, actual)`
  - `TEST_ASSERT_EQUAL_HEX32(expected, actual)`
  - `TEST_ASSERT_EQUAL_HEX64(expected, actual)`
  - `TEST_ASSERT_EQUAL_HEX8_ARRAY(expected, actual, elements)`

• **Numerical Assertions: Bitwise**
  - `TEST_ASSERT_BITS(mask, expected, actual)`

• **Numerical Assertions: Integers**
  - `TEST_ASSERT_TRUE(condition)`
  - `TEST_ASSERT_FALSE(condition)`
  - `TEST_ASSERT(condition)`
  - `TEST_ASSERT_UNLESS(condition)`
  - `TEST_FAIL()`
  - `TEST_FAIL_MESSAGE(message)`

• **Numerical Assertions: Bitwise**
  - `TEST_ASSERT_BITS(mask, expected, actual)`
- TEST_ASSERT_BITS_HIGH(mask, actual)
- TEST_ASSERT_BITS_LOW(mask, actual)
- TEST_ASSERT_BIT_HIGH(mask, actual)
- TEST_ASSERT_BIT_LOW(mask, actual)

- Numerical Assertions: Floats
  - TEST_ASSERT_FLOAT_WITHIN(delta, expected, actual)
  - TEST_ASSERT_EQUAL_FLOAT(expected, actual)
  - TEST_ASSERT_EQUAL_DOUBLE(expected, actual)

- String Assertions
  - TEST_ASSERT_EQUAL_STRING(expected, actual)
  - TEST_ASSERT_EQUAL_STRING_LEN(expected, actual, len)
  - TEST_ASSERT_EQUAL_STRING_MESSAGE(expected, actual, message)
  - TEST_ASSERT_EQUAL_STRING_LEN_MESSAGE(expected, actual, len, message)

- Pointer Assertions
  - TEST_ASSERT_NULL(pointer)
  - TEST_ASSERT_NOT_NULL(pointer)

- Memory Assertions
  - TEST_ASSERT_EQUAL_MEMORY(expected, actual, len)

1.15.9 CLI Guide

1.16 Static Code Analysis

Automated code analysis without hassle!

Static analysis became an important part of software development cycle. It can identify potential bugs, vulnerabilities and security threats by doing an analysis on the source code level without having to test it on hardware or execute any code.

Static Code Analysis helps reduce development cost by enabling engineers to detect the precise location of defects and eliminate issues more efficiently and earlier in the development cycle. It can also ensure compliance with internal or industry coding standards such as MISRA, CERT, etc.
1.16.1 Key features

- Fully integrated within the PlatformIO ecosystem and easy to execute on the entire project.
- Straightforward integration with Continuous Integration services.
- Possibility to reuse the same setup on other projects.
- Easy and flexible rule configuration.
- Comprehensive and detailed error information
- Multiple architectures and development platforms.
- Cross-platform: Windows, MacOS, Linux.

Static Code Analysis can detect a wide range of known defects in C/C++ code, including:

- Potential NULL pointer dereferences
- Possible indexing beyond array bounds
- Suspicious assignments
- Reads of potentially uninitialized objects
- Unused variables or functions
- Out of scope memory usage

**Warning:** Before performing a static analysis check, make sure your project builds without errors. For information about how to build a project, see the `pio run` command or VSCode guide.

1.16.2 User Interface

There is the rich and friendly interface for Static Code Analysis in PlatformIO Home. It allows you to filter messages or directly jump to an issue in a source code.
### Defects Summary

<table>
<thead>
<tr>
<th>Component</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>include</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>include/external</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>src</td>
<td>10</td>
<td>35</td>
<td>5</td>
</tr>
<tr>
<td>src/comms</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>src/hw/spl</td>
<td>4</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>src/hw/uart</td>
<td>1</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>src/misra</td>
<td>26</td>
<td>191</td>
<td>58</td>
</tr>
<tr>
<td>src/samples</td>
<td>24</td>
<td>165</td>
<td>58</td>
</tr>
<tr>
<td>src/sensors</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

**Total**: 65 High / 416 Medium / 127 Low

### Top Defects

<table>
<thead>
<tr>
<th>Level</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH</td>
<td>Array <code>arr[10]</code> accessed at index 10, which is out of bounds.</td>
</tr>
<tr>
<td>HIGH</td>
<td>Memory pointed to by <code>pl</code> is freed twice.</td>
</tr>
<tr>
<td>HIGH</td>
<td>Mismatching allocation and deallocation: <code>pl</code></td>
</tr>
<tr>
<td>HIGH</td>
<td>Memory is allocated but not initialized: <code>pl</code></td>
</tr>
<tr>
<td>HIGH</td>
<td>Resource leak: <code>pMemory</code></td>
</tr>
</tbody>
</table>

If you enjoy using PlatformIO, please star our projects on GitHub!

**STAR PlatformIO Core**
1.16.3 Configuration

Static Code Analysis allows selecting what tool is used for finding defects in the project, what source files are checked. Static Code Analysis can be configured from “platformio.ini” (Project Configuration File) using the next options:
1.16.4 Check tools

You can switch between or specify multiple tools used for finding defects using `check_tool` option:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck, clangtidy
```

Detailed information about supported check tools and their configuration process can be found on these pages:

Cppcheck

Cppcheck is a static analysis tool for C/C++ code. It provides a unique code analysis to detect bugs and focuses on detecting undefined behavior and dangerous coding constructs. The goal is to detect only real errors in the code (i.e. have very few false positives). More information about this tool on the official webpage.

**Hint:** Cppcheck is rarely wrong about reported errors. But there are many bugs that it doesn’t detect. You will find more bugs in your software by testing your software carefully than by using Cppcheck.

Contents

- Features
- Additional checks
- Configuration
- Extra flags
- Suppressing warnings
- Addons (MISRA, CERT)
  - MISRA
  - CERT

Features

Cppcheck supports a wide variety of static checks that may not be covered by the compiler itself. These checks are static analysis checks that can be performed at a source code level. The program is directed towards static analysis checks that are rigorous, rather than heuristic in nature.

Some of the defects that might be detected include:

- Automatic variable checking
- Bounds checking for array overruns
- Classes checking (e.g. unused functions, variable initialization, and memory duplication)
- Usage of deprecated or superseded functions
- Exception safety checking, for example, usage of memory allocation and destructor checks
- Memory leaks, e.g. due to lost scope without deallocation
- Resource leaks, e.g. due to forgetting to close a file handle
- Invalid usage of Standard Template Library functions and idioms
- Miscellaneous stylistic and performance errors

**Additional checks**

Be default **Cppcheck** is configured to check the next additional defects:

- warning
- style
- performance
- portability
- unusedFunction

The full list of supported check with detailed description is located on the official webpage.

**Configuration**

**Cppcheck** is implicitly used as the default check tool when `check_tool` option in "platformio.ini" (Project Configuration File) is not set. To be explicit, you can specify it in the configuration directly:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck
check_flags = --enable=all
```

Useful options that can be used for adjusting check process:

**Extra flags**

Useful flags that can help more precisely configure **Cppcheck** to satisfy your project requirements:
<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--enable=&lt;id&gt;</code></td>
<td>Enable additional checks. The available ids are: all, warning, style,</td>
</tr>
<tr>
<td></td>
<td>performance, portability, information, unusedFunction, missingInclude</td>
</tr>
<tr>
<td><code>--std=&lt;id&gt;</code></td>
<td>Set standard. The available options are: c89, c99, c11, c++03, c++11,</td>
</tr>
<tr>
<td></td>
<td>c++17, c++20 (default)</td>
</tr>
<tr>
<td><code>--language=&lt;language&gt;</code></td>
<td>Forces Cppcheck to check all files as the given language. Valid values are: c, c++</td>
</tr>
<tr>
<td><code>--inline-suppr</code></td>
<td>Enable inline suppressions. Use them by placing one or more comments, like: // cppcheck-suppress warningId on the lines before the warning to suppress (enabled by default if no extra flags specified).</td>
</tr>
<tr>
<td><code>--suppress=&lt;spec&gt;</code></td>
<td>Suppress warnings that match &lt;spec&gt;. The format of &lt;spec&gt; is: [error id]:[filename]:[line]</td>
</tr>
<tr>
<td><code>--platform=&lt;type&gt;</code></td>
<td>Specifies platform-specific types and sizes. The available built-in platforms are: unix32, unix64, win32A, win32W, win64, avr8, native, unspecified (default)</td>
</tr>
<tr>
<td><code>--inconclusive</code></td>
<td>Allow reporting defects even though the analysis is inconclusive.</td>
</tr>
<tr>
<td><code>-D&lt;ID&gt;</code></td>
<td>Define a preprocessor symbol. Example: -DDEBUG=1</td>
</tr>
<tr>
<td><code>-U&lt;ID&gt;</code></td>
<td>Undefine preprocessor symbol. Use -U to explicitly hide certain #ifdef &lt;ID&gt; code paths from checking. Example: -UDEBUG</td>
</tr>
<tr>
<td><code>-I &lt;dir&gt;</code></td>
<td>Give a path to search for include files. Give several -I parameters to give several paths.</td>
</tr>
<tr>
<td><code>-j &lt;jobs&gt;</code></td>
<td>Start &lt;jobs&gt; threads to do the checking simultaneously.</td>
</tr>
</tbody>
</table>

### Suppressing warnings

It might be useful to explicitly instruct `Cppcheck` to ignore some of the known defects in project codebase. Since `--inline-suppr` is enabled by default, it’s possible to directly mark pieces of code that will be excluded from `Cppcheck` report using `// cppcheck-suppress warningId` syntax.

**Note:** Warning ID can be found in square brackets at the end of defect description, for example: `src\Blink.cpp:17: [low:style] The function 'loop' is never used. [unusedFunction]`

By default, Static Code Analysis command doesn’t scan framework sources and that’s why some functions from in your project might be reported as unused. For example, you can ignore warnings about `setup` and `loop` functions from Arduino-based projects:

```cpp
// cppcheck-suppress unusedFunction
void setup()
{
    ...
}

// cppcheck-suppress unusedFunction
void loop()
{
    ...
}
```
Addons (MISRA, CERT)

**Cppcheck** provides several addon scripts that analyze dump files to check compatibility with secure coding standards and to locate various issues. Most useful addons for verifying compliance with popular guidelines are **MISRA** and **CERT**.

**MISRA**

MISRA is a proprietary set of software development guidelines for the C/C++ programming languages developed by MISRA (Motor Industry Software Reliability Association). It aims to facilitate code safety, security, portability, and reliability in the context of embedded systems, specifically those systems programmed in ISO C/C++.

**Note:** Since this standard is proprietary, **Cppcheck** does not display error text by specifying only the number of violated rules (for example, [c2012-21.3]). If you want to display full texts for violated rules, you will need to create a text file containing MISRA rules, which you will have to pass when calling the script with **--rule-texts** flag.

In order to use MISRA addon you will need to provide a special file with the description of MISRA rules. Usually, it has the next contents:

```
Appendix A Summary of guidelines
Rule 3.1 Required
R3.1 Rule description
Rule 4.1 Required
...
Rule 21.3 Required
R21.3 Rule description
Rule 21.4
R21.4 Rule description
```

Next, you need to instruct **Cppcheck** that you want to run an additional addon script. Since this script requires an additional file with rules, you can pass it via a special **json** file:

```
{
  "script": "addons/misra.py",
  "args": ["--rule-texts=misra-rules.txt"]
}
```

Finally, add new flag to **check_flags**:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck
check_flags =
  cppcheck: --addon=misra.json
```

The full list of implemented MISRA checks can be found on the official webpage.

**CERT**

SEI CERT coding standard provides rules for secure coding in the C programming language. The goal of these rules and recommendations is to develop safe, reliable, and secure systems, for example by eliminating undefined behaviors that can lead to undefined program behaviors and exploitable vulnerabilities.
In order to use the CERT addon, simply specify it as an additional flag in `check_flags` section:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck
check_flags =
    cppcheck: --addon=cert.py
```

**Clang-Tidy**

Clang-Tidy is a clang-based C++ “linter” tool. Its purpose is to provide an extensible framework for diagnosing and fixing typical programming errors, like style violations, interface misuse, or bugs that can be deduced via static analysis. Official page can be found here.

### Contents

- Features
- Configuration
- Supported checks
- Extra flags

**Features**

Clang-Tidy supports a large variety of static checks that may not be covered by the compiler itself. These checks are static analysis checks that can be performed at a source code level.

Some of the defects that might be detected include:

- Buffer overflow
- Potential NULL pointer dereferences
- Use of memory that has already been deallocated
- Out of scope memory usage
- Failure to set a return value from a subroutine

**Configuration**

To enable Clang-Tidy tool simply add it to the `check_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
check_tool = clangtidy
```

Useful options that can be used used for adjusting check process:
Supported checks

There are currently the following groups of most used checks (By default all checks are enabled):

<table>
<thead>
<tr>
<th>Check</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abseil-*</td>
<td>Checks related to Abseil library.</td>
</tr>
<tr>
<td>boost-*</td>
<td>Checks related to Boost library.</td>
</tr>
<tr>
<td>bugprone-*</td>
<td>Checks that target bugprone code constructs.</td>
</tr>
<tr>
<td>cert-*</td>
<td>Checks related to CERT Secure Coding Guidelines.</td>
</tr>
<tr>
<td>cppcoreguidelines-*</td>
<td>Checks related to C++ Core Guidelines.</td>
</tr>
<tr>
<td>clang-analyzer-*</td>
<td>Clang Static Analyzer checks.</td>
</tr>
<tr>
<td>google-*</td>
<td>Checks related to Google coding conventions.</td>
</tr>
<tr>
<td>hicpp-*</td>
<td>Checks related to High Integrity C++ Coding Standard.</td>
</tr>
<tr>
<td>modernize-*</td>
<td>Checks that advocate usage of modern (currently modern means C++11) language constructs.</td>
</tr>
<tr>
<td>performance-*</td>
<td>Checks that target performance-related issues.</td>
</tr>
<tr>
<td>portability-*</td>
<td>Checks that target portability-related issues that don’t relate to any particular coding style.</td>
</tr>
<tr>
<td>readability-*</td>
<td>Checks that target readability-related issues that don’t relate to any particular coding style.</td>
</tr>
</tbody>
</table>

The full list of supported checks can be found on the official webpage.

Extra flags

Useful flags that can help more precisely configure Clang-Tidy to satisfy your project requirements:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>--checks=&lt;string&gt;</td>
<td>Comma-separated list of enabled checks (*) default)</td>
</tr>
<tr>
<td>--fix</td>
<td>Apply suggested fixes. Without --fix-errors clang-tidy will bail out if any compilation errors were found.</td>
</tr>
<tr>
<td>--fix-errors</td>
<td>Apply suggested fixes even if compilation errors were found. If compiler errors have attached fix-its, clang-tidy will apply them as well.</td>
</tr>
<tr>
<td>--format-style=&lt;string&gt;</td>
<td>Style for formatting code around applied fixes: llvm, google, webkit, mozilla, none (default)</td>
</tr>
<tr>
<td>--system-headers</td>
<td>Display the errors from system headers.</td>
</tr>
</tbody>
</table>

An example with enabling specific checks and fixing code on the fly:

```plaintext
[env:myenv]
platform = ...
board = ...
check_tool = clangtidy
check_flags =
   clangtidy: --checks=-*,cert-*,clang-analyzer-* --fix
```

PVS-Studio

PVS-Studio is a static code analysis tool for detecting bugs and security weaknesses in the source code of programs, written in C, C++, C# and Java. It analyze source code intended for 32-bit, 64-bit and embedded ARM platforms.
Official page can be found here.

### Contents

- Features
- Configuration
- Extra flags
- Obtaining license

### Features

**PVS-Studio** performs a wide range of code checks, and it is also useful in finding misprints and Copy-Paste errors. These checks are static analysis checks that can be performed at a source code level. Some of the defects that might be detected include:

- Arithmetic over/underflow
- Array index out of bounds
- Undefined/unspecified behavior
- Incorrect usage of exceptions
- Buffer overrun
- Null pointer/null reference dereference
- Improper understanding of function/class operation logic
- Illegal bitwise/shift operations

The full list of supported checks can be found on the official webpage.

### Configuration

To enable **PVS-Studio** tool simply add it to the `check_tool` option in the `platformio.ini` (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
check_tool = pvs-studio
```

Useful options that can be used for adjusting check process:

### Extra flags

Useful flags that can help more precisely configure **PVS-Studio** to satisfy your project requirements:
Flag | Meaning
--- | ---
--analysis-mode <arg> | Analysis mode (0 - full analysis (default), 1 - 64-bit. analysis, 2 - reserved, 4 - general analysis, 8 - optimization, 16 - customer’s specific requests, 32 - MISRA)
--analyzer-errors <arg> | Errors activation (Default: all errors is on)
--errors-off <arg> | Errors OFF (Default: all errors is on)
--exclude-path <arg> | All code that is located under the path will be excluded from analysis
--lic-file <arg> | Path to custom license file. Default locations ~/.config/PVS-Studio/PVS-Studio.lic on Unix and %APPDATA%\PVS-Studio\PVS-Studio.lic on Windows
--rules-config <arg> | Specifies the path to rules configuration file.
--platform <arg> | Platform name (Win32, x64, etc) (Default: ARM)

An example with a special analysis mode, disabled errors and license file:
```
[env:myenv]
platform = ...
board = ...
check_tool = pvs-studio
check_flags =
    pvs-studio: --analysis-mode=4 --errors-off=V532,V586 --lic-file=/path/to/file.lic
```

**Obtaining license**

Since **PVS-Studio** is a paid B2B solution, a license should be purchased. But PVS-Studio can be used for free of charge, for example for checking open source projects. More information about the cases when you can get a free **PVS-Studio** license can be found on the official webpage.

**Tip:** If you’re experiencing problems with the license file or see the following error message: License information is incorrect. Please check your registration data or contact Customer Support, try saving the license file in UTF-8 + BOM format with the following contents:

```
n@domain.com
AAAA-BBBB-CCCC-DDDD
```

### 1.16.5 Defect severity

Defect severity is a classification of software defect (bug, vulnerability, etc) that indicates the degree of negative impact on the quality of software. **Static Code Analysis** uses the next classification of possible defects:

<table>
<thead>
<tr>
<th>Severity</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>Issues that are possibly bugs</td>
</tr>
<tr>
<td>medium</td>
<td>Suggestions about defensive programming in order to prevent potential bugs</td>
</tr>
<tr>
<td>low</td>
<td>Issues related to code cleanup and performance (unused functions, redundant code, const-ness, etc)</td>
</tr>
</tbody>
</table>

### 1.16. Static Code Analysis
1.16.6 CLI Guide

Static Code Analysis can be configured using command line commands. Detailed description of these commands can be found here:

1.17 Remote Development

Your devices are always with you!

PlatformIO Remote Development Solution allows you to work remotely with devices from Anywhere In The World. No matter where you are now! Run a small and cross-platform PlatformIO Remote Agent on a remote machine and you are able to list active devices (wireless + wired), to upload firmware (program), to process remote unit tests, or to start remote debugging session via Remote Serial Port Monitor.

Using PlatformIO Remote Development Solution you can share your devices with colleagues across your organization or friends. In combination with Cloud IDE, you can create awesome things at any time when inspiration comes to you.

You should have PlatformIO Account to use PlatformIO Remote Development Solution. A registration is FREE.

1.17.1 Features

- Remote Device Manager
- Remote Serial Port Monitor
- Remote Firmware Updates
- Share devices with your team members
- Continuous Deployment
- Continuous Delivery
- Remote Unit Testing

1.17.2 Use Cases

Cloud IDE Program your devices from anywhere in the world using the most popular Cloud IDE. You do not need to install any extra software, no need to have static IP or open network ports. Everything works out of the box.
Devices behind card sized PC Work with your favorite development environment and program devices connected to card-sized PC (Raspberry Pi, Cubie Board, etc.). You do not need to open SSH ports, install any extra Linux packages, toolchains.

Remote Unit Testing Instruct any of Continuous Integration services to run remote tests on a physical device. See the documentation for Remote Test Runner.

How does it work?

- You commit new changes to your source code repository
- Continuous Integration service deploys unit tests to a remote agent
- Unit Testing engine runs tests on a physical device, process them, and send results
- Continuous Integration service prints results in human readable format
- If one test fails, current CI build will fail too.

Board Farm A similar concept as described in “Remote Unit Testing” above. Let’s imagine that you need to test some logic on the unlimited number of target devices. Very often it can be the same hardware prototype but with different factory revisions.

You connect these devices via USB hub to PC and instruct PlatformIO Remote Development Solution to process your test on ALL targets connected to a specific agent. See documentation below.

Remote Serial Monitor Sometimes you don’t have physical access to a target device but you need to read data from some serial port. PlatformIO Remote Development Solution allows you to connect to a remote agent and list connected devices with their serial ports. See pio remote device monitor command for details.
1.17.3 Technology

![PIO Remote™ Architecture Diagram]

- **Cloud IDE Integration**
- **Remote Unit Testing**
- **Continuous Deployment**

**Remote Solutions**
- Device Manager
- Firmware (Program) Updates
- Serial Port Monitor
- Standard Streams Monitor

**PIO Agent**
- PC, Card-sized PC (Raspberry Pi, etc.)
  - Project Synchronization
  - Asynchronous Command Processor
  - Process Control System

**PIO Client**
- PC, Browser, SDK, etc.
  - Project Synchronization
  - Multiagent Command Processor
  - Asynchronous Result Processor

**PIO Cloud**
- Broker

**PIO Remote™**
- Connection Protocol
- Authentication Protocol
- Transport Protocol

**TCP/IP Stack**

**Links**
- **USB Link**: Wired devices
- **Ethernet Link**: Wired devices
- **Wi-Fi Link**: Wireless devices
PlatformIO Remote Development Solution is an own PlatformIO technology for remote solutions without external dependencies to operating system or its software based on client-server architecture. The Server component (PlatformIO Cloud) plays a role of coupling link between PlatformIO Remote Agent and Client (PlatformIO Remote CLI, Cloud IDE, Continuous Integration, SDKs, etc.). When you start PlatformIO Remote Agent, it connects over the Internet with PlatformIO Cloud and listen for the actions/commands which you can send in Client role from anywhere in the world.

PlatformIO Remote Development Solution is multi-agents and multi-clients system. A single agent can be shared with multiple clients, where different clients can use the same agent. This approach allows one to work with distributed hardware located in the different places, networks, etc.

This technology allows one to work with remote devices in generic form as you do that with local devices using PlatformIO ecosystem. The only one difference is a prefix “remote” before each generic PlatformIO command. For example, listing of local and remote devices will look like pio device list and pio remote device list.

1.17.4 Installation

PlatformIO Remote Development Solution is built into PlatformIO IDE. Please open PlatformIO IDE Terminal and run pio remote --help command for usage (see PlatformIO Remote CLI).

If you do not have PlatformIO IDE, or use Cloud IDE or a card-sized PC (Raspberry Pi, BeagleBoard, etc.), please install PlatformIO Core (CLI).

1.17.5 Quick Start

1. Start PlatformIO Remote Development Solution Agent using pio remote agent start command on a remote machine where devices are connected physically or are accessible via network. PlatformIO Remote Development Solution Agent works on Windows, macOS, Linux and Linux ARMv6+. It means that you can use desktop machine, laptop or credit card sized PC (Raspberry Pi, BeagleBoard, etc).

   You can share own devices/hardware with friends, team or other developers using pio remote agent start --share option.

2. Using host machine (PlatformIO Remote CLI, Cloud IDE Terminal in a browser, SDKs, etc.), please authorize via pio account login command with the same credentials that you used on the previous step. Now, you can use PlatformIO Remote CLI commands to work with remote machine and its devices.

   You don’t have to networking or other access to remote machine where PlatformIO Remote Development Solution Agent is started.

   If you use PlatformIO Remote Development Solution in pair with Continuous Integration or want automatically authorize, please set PLATFORMIO_AUTH_TOKEN system environment variable instead of using pio account login command.

Note: In case with Cloud IDE, your browser with Cloud IDE’s VM is a “host machine”. The machine where devices are connected physically (your real PC) is called “remote machine” in this case. You should run PlatformIO Remote Development Solution Agent here (not in Cloud IDE’s Terminal).

Note: Please use local IP as “upload port” when device is not connected directly to a remote machine where PlatformIO Remote Development Solution Agent is started but supports natively Over-the-Air (OTA) updates. For example, Espressif 8266 and Over-the-Air (OTA) update. In this case, the final command for remote OTA update will look as pio remote run -t upload --upload-port 192.168.0.255 or pio remote run -t upload --upload-port myesp8266.local.
1.17.6 CLI Guide

1.18 PlatformIO Account

PlatformIO Account is required for:
- Community Forum
- Remote Development
- Managing organizations and owners of an organization
- Managing teams and team memberships
- Managing packages in the registry
- Managing resource (package) access.

1.18.1 PlatformIO IDE

PlatformIO IDE has built-in UI in PlatformIO Home to manage PlatformIO Account. You can create a new account, reset your password, update profile, or fetch an authentication token. This UI must be used to login with OAuth.

1.18.2 CLI Guide

1.19 Cloud & Desktop IDE

1.19.1 Native Extensions

Native extensions and plugins provide official integration with IDEs/Text Editors and contain built-in PlatformIO Core (CLI) and PlatformIO Home. We highly recommend to use the next native integrations for:
- VSCode
- CLion (Experimental)

Note: In our experience, VSCode offers better system performance, and users have found it easier to get started

1.19.2 Cloud IDE

Cloud9

Cloud9 combines a powerful online code editor with a full Ubuntu workspace in the cloud. Workspaces are powered by Docker Ubuntu containers that give you full freedom over your environment, including sudo rights. Do a git push, compile SASS, see server output, and Run apps easily with the built-in Terminal and Runners.

Contents
- Cloud9
  - Demo
Note:

1. Please make sure to read Remote Development guide first.
2. You need PlatformIO Account if you don’t have it. Registration is FREE.
3. You should have a running PlatformIO Remote Agent on a remote machine where hardware devices are connected physically or accessible for the remote operations. See Remote Development Quick Start for details.

Demo
Integration

1. Sign in to Cloud9. A registration is FREE and gives you for FREE 1 private workspace (where you can host multiple PlatformIO Projects) and unlimited public workspaces.

2. Create a new workspace using Blank template

Create a new workspace

<table>
<thead>
<tr>
<th>Workspace name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>platformio-workspace</td>
<td>Workspace for private PlatformIO Projects</td>
</tr>
</tbody>
</table>

Hosted workspace | Clone workspace | Remote SSH workspace | Salesforce

- **Private**
  - This is a workspace for your eyes only

- **Public**
  - This will create a workspace for everybody to see

Clone from Git or Mercurial URL (optional)

```
e.g. ajaxorg/ace or git@github.com:ajaxorg/ace.git
```

Choose a template

- HTML5
- Node.js
- PHP
- Python
- Django
- Ruby
- C++
- Wordpress
- Rails Tutorial
- Blank
- Harvard’s CS50

Create workspace

3. Install PlatformIO Core (CLI) using Cloud IDE Terminal. Paste a next command

```
sudo python -c "$(curl -fsSL https://raw.githubusercontent.com/platformio/platformio/develop/scripts/get-platformio.py)"
```
4. Log in to PlatformIO Account using `pio account login` command.

**Quick Start**

Let’s create our first PlatformIO-based Cloud9 Project

1. Initialize new PlatformIO-based Project. Run a next command in Cloud IDE Terminal:

   ```
   pio project init --board <ID>
   # initialize project for Arduino Uno
   pio project init --board uno
   ```

   To get board ID please use `pio boards` command or Embedded Boards Explorer.

2. Create new source file named `main.cpp` in `src` folder using Project Tree (left side). Please make right click on `src` folder, then “New File” and insert a next content:

   ```
   #include <Arduino.h>

   int i = 0;

   void setup() {
     Serial.begin(9600);
     Serial.println("Hello Cloud9!");
   }
   ```

   (continues on next page)
```c
void loop() {
    /* serial echo */
    while (Serial.available()) {
        Serial.write(Serial.read());
    }
    i++;
    Serial.println(i);
    delay(100);
}
```

3. If you prefer to work with PlatformIO Core (CLI) CLI, then you can process project using Cloud IDE Terminal and the next commands:

   - `pio run` - build project locally (using Cloud IDE’s virtual machine)
   - `pio run -t clean` - clean project
   - `pio remote run -t upload` - upload firmware (program) to a remote device
   - `pio remote device list` - list available remote devices
PlatformIO Build System

Cloud9 allows one to create own build system and use hotkey or command (Menu: Run > Build) to build a project.

Let’s create PlatformIO Build System that will be used for C/C++/H/INO/PDE files by default. Please click on Menu: Run > Build System > New Build System and replace all content with the next:

```json
{
    "cmd" : ["pio", "run", "-d", "$file"],
    "info" : "Building $project_path/$file_name",
    "selector": "^.*\.(cpp|c|h|hpp|S|ini|ino|pde)$"
}
```

Save new Build System and give a name PIOBuilder. Now, you can select it as default Build System using Menu: Run > Build System > PIOBuilder.

Remote Device Manager

Remote Device Manager works in pair with Remote Development. You can list remote devices that are connected to host machine where PlatformIO Remote Agent is started or are visible for it.

Let’s create New Run Configuration (shortcut) that will be used for Remote Device Manager. Please click on Menu: Run > Run Configurations > Manage..., then “Add New Config” and specify the next values:

- **First Blank Input**: a name of runner. Please set it to “PIO: Remote Devices”
- **Command**: set to pio remote device list
- **Runner**: set to “Shell command”
Remote Firmware Uploading

Remote Firmware Uploading works in pair with Remote Development. You can deploy firmware (program) to any devices which are visible for PlatformIO Remote Agent.

Let’s create New Run Configuration (shortcut) that will be used for Remote Firmware Uploading. Please click on Menu: Run > Run Configurations > Manage..., then “Add New Config” and specify the next values:

- **First Blank Input**: a name of runner. Please set it to “PIO: Remote Upload”
- **Command**: set to `pio remote run -t upload`
- **Runner**: set to “Shell command”
PlatformIO Documentation, Release 5.0.5a1

2614 Chapter 1. Contents

PlatformIO Plus (https://pioplus.com) v0.3.1
Building project locally

Verbose mode can be enabled via `-v, --verbose` option
Collected 25 compatible libraries
Looking for dependencies...
Project does not have dependencies
Checking program size

Verbose mode can be enabled via `-v, --verbose` option
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA1411
Uploading .pioenvs/uno/firmware.hex

avrdude: AVR device initialized and ready to accept instructions
Reading | #%%%%%%%%%%%%%%%%%%%%%%%%%%%% % | 100% 0.005s
avrdude: Device signature = 0xe0950f
avrdude: reading input file "pioenvs/uno/firmware.hex"
avrdude: writing flash (2372 bytes):

Writing | #%%%%%%%%%%%%%%%%%%%%%%%%%%%% % | 100% 0.391s
avrdude: 2372 bytes of flash written
avrdude: verifying flash memory against .pioenvs/uno/firmware.hex:
avrdude: load data flash data from input file .pioenvs/uno/firmware.hex:
avrdude: input file .pioenvs/uno/firmware.hex contains 2372 bytes
avrdude: reading on-chip flash data:
Reading | #%%%%%%%%%%%%%%%%%%%%%%%%%%%% % | 100% 0.315s
avrdude: verifying ...
avrdude: 2372 bytes of flash verified

avrdude done. Thank you.
============================================================================= [SUCCESS] Took 3.109 seconds
Remote Serial Port Monitor

Remote Serial Port Monitor works in pair with Remote Development. You can read or send data to any device that is connected to host machine where PlatformIO Remote Agent is started. To list active agents please use this command `pio remote agent list`.

Let’s create New Run Configuration (shortcut) that will be used for Remote Serial Port Monitor. Please click on Menu: Run > Run Configurations > Manage..., then “Add New Config” and specify the next values:

- **First Blank Input**: a name of runner. Please set it to “PIO: Remote Serial Monitor”
- **Command**: set to `pio remote device monitor`
- **Runner**: set to “Shell command”

Multi-Project workspace

You can have multiple PlatformIO-based Projects in the same workspace. We recommend a next folders structure:

```
project-A
  lib
    README
    platformio.ini
  src
```

(continues on next page)
In this case, you need to create 2 “New Run Configuration” for Remote Firmware Uploading with using the next commands:

- `pio remote run --project-dir project-A -t upload` for Project-A
- `pio remote run -d project-B -t upload` for Project-B

See documentation for `pio remote run --project-dir` option.

**Codeanywhere**

Codeanywhere is a Cross Platform Cloud IDE and it has all the features of Desktop IDE but with additional features only a cloud application can give you! Codeanywhere is very flexible and you can set up your workflow any way you want it. The elegant development environment will let you focus on building great applications quicker. All the features you will need for any coding task are built into Codeanywhere, making development more productive and fun.

**Contents**

- **Codeanywhere**
  - Demo
  - Integration
  - Quick Start
  - Run Button
  - Remote Device Manager
  - Remote Firmware Uploading
  - Remote Serial Port Monitor
  - Multi-Project workspace

**Note:**

1. Please make sure to read Remote Development guide first.
2. You need PlatformIO Account if you don’t have it. Registration is FREE.
3. You should have a running PlatformIO Remote Agent on a remote machine where hardware devices are connected physically or accessible for the remote operations. See Remote Development Quick Start for details.
Demo

1. Sign in to Codeanywhere. A registration is FREE and gives you unlimited private projects within the one Container.

2. Open Dashboard Projects

3. Create a new Project and open it. In Connection Wizard create new Container:
   - **Name** set to “PlatformIO”
   - **Stack** search for Python stack (not Python3) that is based on Ubuntu OS.
   - Click on “Create” button.

Integration

1. **Sign in to Codeanywhere.** A registration is FREE and gives you unlimited private projects within the one Container.

2. **Open Dashboard Projects**

3. Create a new Project and open it. In Connection Wizard create new Container:
   - **Name** set to “PlatformIO”
   - **Stack** search for Python stack (not Python3) that is based on Ubuntu OS.
   - Click on “Create” button.
4. Open **SSH-Terminal** tab (right click on **Container (PlatformIO)** > **SSH Terminal**) and install **PlatformIO Core (CLI)** using a next command

```
sudo python -c "$(curl -fsSL https://raw.githubusercontent.com/platformio/platformio/develop/scripts/get-platformio.py)"
```
5. Log in to PlatformIO Account using pio account login command.

Quick Start

Let’s create our first PlatformIO-based Codeanywhere Project

1. Initialize new PlatformIO-based Project. Run a next command in a Cloud IDE SSH Terminal:

   ```
   pio project init --board <ID>
   # initialize project for Arduino Uno
   pio project init --board uno
   ```

   To get board ID please use pio boards command or Embedded Boards Explorer.

   If you do not see created project, please refresh Project Tree using right-click on Container Name (PlatformIO) > Refresh.

2. Create new source file named main.cpp in src folder using Project Tree (left side). Please make right click on src folder, then “Create File” and insert a next content:

   ```
   #include <Arduino.h>
   int i = 0;
   void setup() {
   ```

(continues on next page)
Serial.begin(9600);
Serial.println("Hello Codeanywhere!");

void loop() {
    /* serial echo */
    while (Serial.available()) {
        Serial.write(Serial.read());
    }
    i++;
    Serial.println(i);
    delay(100);
}

3. If you prefer to work with PlatformIO Core (CLI) CLI, then you can process project using Cloud IDE SSH Terminal and the next commands:
• *pio run* - build project locally (using Cloud IDE’s virtual machine)
• *pio run -t clean* - clean project
• *pio remote run -t upload* - upload firmware (program) to a remote device
• *pio remote device list* - list available remote devices
• *pio remote device monitor* - Remote Serial Port Monitor

4. We recommend to hide “Hidden Files”. You can do that via Cloud IDE Menu: View > Show Hidden Files.

### Run Button

Codeanywhere provides a quick “Run Project” button where you can specify own command. Let’s add “PlatformIO Build Project” command:

1. Open “Project Config” via right click on Container Name (PlatformIO) > Config
2. Set commands field to

   ```json
   "commands": [
     "pio run"
   ]
   ```


Now, try to click on “Run Project” button. You can assign any PlatformIO command to this button.
```json
"run": [
  {
    "name": "default",
    "default": "true",
    // current working dir
    "cwd": "~/workspace",
    // run commands list
    "commands": [
      "pio run"
    ],
    // environment variables
    // example:
    "environment": {
      // "PYTHONPATH": "~/Python"
    },
    // if "preview" is set, it will open the link in a
    // preview panel after setting environment variables
    // and all commands have been executed
    "preview": {
      "url": "http://{{DOMAIN}}:{{PORT}}/",
      "type": "external"
    },
    "find_in_files": {
      "ignore": [
        // file types to ignore in search
        ".git", ".svn"
      ]
    }
  }
]```
Remote Device Manager

Remote Device Manager works in pair with *Remote Development*. You can list remote devices that are connected to host machine where *PlatformIO Remote Agent* is started or are visible for it.

1. Open Cloud IDE SSH Terminal
2. Paste this command

```bash
pio remote device list
```

Remote Firmware Uploading

Remote Firmware Uploading works in pair with *Remote Development*. You can deploy firmware to any devices which are visible for *PlatformIO Remote Agent*.

1. Open Cloud IDE SSH Terminal
2. Paste this command

```bash
pio remote run -t upload
```
Remote Serial Port Monitor

Remote Serial Port Monitor works in pair with Remote Development. You can read or send data to any device that is connected to host machine where PlatformIO Remote Agent is started. To list active agents please use this command `pio remote agent list`.

1. Open Cloud IDE SSH Terminal
2. Paste this command

```bash
pio remote device monitor
```

Verbose mode can be enabled via `-v` option

Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA1411
Uploading .pioenvs/uno/firmware.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ############################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f
avrdude: reading input file "pioenvs/uno/firmware.hex"
avrdude: writing flash (2578 bytes):

Writing | ############################################################## | 100% 0.39s

avrdude: 2578 bytes of flash written
avrdude: verifying flash memory against .pioenvs/uno/firmware.hex:
avrdude: load data flash data from input file .pioenvs/uno/firmware.hex:
avrdude: input file .pioenvs/uno/firmware.hex contains 2578 bytes
avrdude: reading on-chip flash data:

Reading | ############################################################## | 100% 0.31s

avrdude: verifying ...
avrdude: 2578 bytes of flash verified

avrdude done. Thank you.
Multi-Project workspace

You can have multiple PlatformIO-based Projects in the same workspace. We recommend a next folders structure:

```
- project-A
  - lib
    - README
  - platformio.ini
  - src
    - main.ino
- project-B
  - lib
    - README
  - platformio.ini
  - src
    - main.cpp
    - main.h
```

In this case, you need to use `-d, --project-dir` option for `pio run` or `pio remote run` commands:

- `pio remote run --project-dir project-A -t upload` build Project-A

- `pio remote run --project-dir project-A -t upload remote firmware uploading` using Project-A

- `pio remote run -d project-B -t upload remote firmware (program) uploading` using Project-B

See documentation for `pio remote run --project-dir` option.
Eclipse Che

Eclipse Che is an open-source Java based developer workspace server and cloud integrated development environment (IDE) which provides a remote development platform for multi-user purpose. The workspace server comes with a RESTful webservice and provides high flexibility. It also contains a SDK which can be used to create plug-ins for languages, frameworks or tools.

Contents

- Eclipse Che
  - Demo
  - Integration
  - Quick Start
  - Multi-Project workspace

Note:

1. Please make sure to read Remote Development guide first.
2. You need PlatformIO Account if you don’t have it. Registration is FREE.
3. You should have a running PlatformIO Remote Agent on a remote machine where hardware devices are connected physically or accessible for the remote operations. See Remote Development Quick Start for details.
Demo

Integration

1. Sign in to Codenvy (based on Eclipse Che). A registration is FREE and gives you unlimited private projects.
2. Open Workspaces tab
3. Click on “Add Workspace”, then switch to “Runtime” tab.
   - **Name** set to “PlatformIO”
   - **Stack** search for **PLATFORMIO**
   - Click on “Create” button, then “Open”.

4. Using opened Terminal, please log in to **PlatformIO Account** using `pio account login` command.

**Quick Start**

Let’s create our first PlatformIO-based Codenvy Project

1. Click on Menu: **Workspace > Create New Project** and select `platformio-arduino-blink` sample. Set “Name” to “Arduino Blink” and press “Create”.
2. Now you can use dropdown Commands menu and process project with “run” command
3. If you prefer to work with *PlatformIO Core (CLI)* CLI, then you can process project using Cloud IDE Terminal and the next commands:

   - `pio run` - build project locally (using Cloud IDE’s virtual machine)
   - `pio run -t clean` - clean project
- `pio remote run -t upload` - upload firmware (program) to a remote device
- `pio remote device list` - list available remote devices
- `pio remote device monitor` - Remote Serial Port Monitor

### Multi-Project workspace

You can have multiple PlatformIO-based Projects in the same workspace. We recommend a next folders structure:

```
- project-A
  - lib
    - README
    - platformio.ini
  - src
    - main.ino

- project-B
  - lib
    - README
    - platformio.ini
  - src
    - main.cpp
    - main.h
```

In this case, you need to use `-d, --project-dir` option for `pio run` or `pio remote run` commands:

- `pio remote run --project-dir project-A -t upload` build Project-A
- `pio remote run --project-dir project-A -t upload` remote firmware uploading using Project-A
- `pio remote run -d project-B -t upload` remote firmware (program) uploading using Project-B

See documentation for `pio remote run --project-dir` option.

### 1.19.3 Desktop IDE

**PlatformIO IDE for Atom**

PlatformIO IDE is the next-generation integrated development environment for IoT.

- Cross-platform build system without external dependencies to the OS software:
  - 800+ **Boards**
  - 35+ **Development Platforms**
  - 20+ **Frameworks**
- **Debugging**
- **Remote Development**
- **Unit Testing**
- C/C++ Intelligent Code Completion
- C/C++ Smart Code Linter for rapid professional development
- Library Manager for the hundreds popular libraries
• Multi-projects workflow with multiple panes
• Themes support with dark and light colors
• Serial Port Monitor
• Built-in Terminal with PlatformIO Core (CLI) and CLI tool (pio, platformio)
• Built-in PlatformIO Home.

Atom is a text editor that’s modern, approachable, yet hackable to the core—a tool you can customize to do anything but also use productively without ever touching a config file.
Installation

Note: Please note that you do not need to install PlatformIO Core (CLI) separately if you are going to use PlatformIO IDE for Atom. PlatformIO Core (CLI) is built into PlatformIO IDE and you will be able to use it within PlatformIO IDE Terminal.

Also, PlatformIO IDE allows one to install PlatformIO Core (CLI) Shell Commands (pio, platformio) globally to your system via Menu: PlatformIO > Install Shell Commands.
I. Atom

0. **Download** and install GitHub’s official Atom text editor. PlatformIO IDE is built on top of it.

1. **Open** Atom Package Manager
   - **Mac OS X**, Menu: Atom > Preferences > Install
   - **Windows**, Menu: File > Settings > Install
   - **Linux**, Menu: Edit > Preferences > Install

2. **Search** for the official `platformio-ide` package

3. **Install** PlatformIO IDE.

II. Clang for Intelligent Code Completion

PlatformIO IDE uses Clang for the Intelligent Code Completion. To check that `clang` is available in your system, please open Terminal and run `clang --version`. If `clang` is not installed, then **install it and restart Atom**:

- **Mac OS X**: Install the latest Xcode along with the latest Command Line Tools (they are installed automatically when you run `clang` in Terminal for the first time, or manually by running `xcode-select --install`)

- **Windows**: Download Clang 3.9.1 for Windows. Please select “Add LLVM to the system PATH” option on the installation step.
  - Clang 3.9.1 for Windows (32-bit)
  - Clang 3.9.1 for Windows (64-bit)
Warning: PLEASE DO NOT INSTALL CLANG 4.0. TEMPORARILY, WE SUPPORT ONLY CLANG 3.9.

If you see a Failed to find MSBuild toolsets directory error in the installation console, please ignore it and press any key to close this window. PlatformIO IDE uses only the Clang completion engine, which should work after that without any problems.

- **Linux:** Using package managers: `apt-get install clang` or `yum install clang`.
- **Other Systems:** Download the latest Clang for the other systems.

Warning: If some libraries are not visible in PlatformIO IDE for Atom and Code Completion or Code Linting does not work properly, please perform Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter)

Quick Start

This tutorial introduces you to the basics of PlatformIO IDE workflow and shows you the creation process for a simple “Blink” example. After finishing, you will have a general understanding of how to work with projects in the IDE.

Launch

After installation, launch PlatformIO IDE by opening Atom. Once Atom is opened, the PlatformIO IDE auto installer will continue to install dependent packages and PlatformIO Core (CLI). Please be patient and let the installation
complete. Once finished, PlatformIO IDE will ask you to reload the Atom window to apply installed components. Please click on Reload Now. After that, PlatformIO IDE is ready for use. Happy coding!

Setting Up the Project

1. Click on the “PlatformIO Home” button on the PlatformIO Toolbar

2. Click on “New Project”, select a board and create a new PlatformIO Project
3. Open the main.cpp file in the src folder and replace its contents with the following:

```c
/**
 * Blink
 *
 * Turns on an LED on for one second,
 * then off for one second, repeatedly.
 */
```

**Warning:** The code below only works with Arduino-based boards. Please visit the PlatformIO Project Examples repository for other pre-configured projects.
```c
#include "Arduino.h"

// Set LED_BUILTIN if it is not defined by Arduino framework
#define LED_BUILTIN 13

void setup()
{
  // initialize LED digital pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_BUILTIN, HIGH);

  // wait for a second
  delay(1000);

  // turn the LED off by making the voltage LOW
  digitalWrite(LED_BUILTIN, LOW);

  // wait for a second
  delay(1000);
}
```
Process Project

PlatformIO IDE proposes different ways to process the project (build, clean, upload firmware, run other targets) using:

- **PlatformIO Toolbar**
- **Menu item PlatformIO**
- **Building / Uploading / Targets** and hotkeys

```cpp
/**
 * Blink
 *
 * Turns on an LED on for one second,
 * then off for one second, repeatedly.
 */

#include "Arduino.h"

// Set LED_BUILTIN if it is not defined by Arduino
// #define LED_BUILTIN 13

void setup()
{
  // initialize LED digital pin as an output
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  // turn the LED on (HIGH is the voltage level)
  digitalWrite(LED_BUILTIN, HIGH);

  // wait for a second
  delay(1000);

  // turn the LED off by making the voltage low
  digitalWrite(LED_BUILTIN, LOW);
}
```
5. Run **Build** and you should see a green “success” result in the build panel:
To upload firmware to the board, run Upload.

6. What is more, you can run specific target or process project environment using Menu: PlatformIO > Run other target... or call targets list from the status bar (bottom, left corner):
And select desired target:
7. To launch the built-in terminal interface, choose Menu: PlatformIO > Terminal or press the corresponding icon in the PlatformIO toolbar:
This provides you fast access to a set of powerful PlatformIO Core (CLI) CLI commands:
8. To run the built-in “Serial Monitor”, choose Menu: PlatformIO > Serial Monitor or press the corresponding icon in the PlatformIO toolbar:
The monitor has several settings to adjust your connection:
It also allows you to communicate with your board in an easy way:
Menu item PlatformIO

platformio-ide package adds to Atom new menu item named Menu: PlatformIO (after Menu: Help item).
PlatformIO Toolbar

PlatformIO IDE Toolbar contains quick access buttons for the popular commands. Each button contains a hint (leave the mouse on it for a moment).

- **PlatformIO Home**
- PlatformIO: Build
- PlatformIO: Upload
- PlatformIO: Clean
- **Debugging**
• Run other target (Build environments, Unit Testing)
• Toggle build panel
• ||
• Find in Project…
• PlatformIO Terminal
• Serial Monitor
• ||
• Atom Settings

Building / Uploading / Targets

• cmd-alt-b/ctrl-alt-b/f9 builds project without auto-uploading.
• cmd-alt-u/ctrl-alt-u builds and uploads (if no errors).
• cmd-alt-c/ctrl-alt-c cleans compiled objects.
• cmd-alt-t/ctrl-alt-t/f7 run other targets (Upload using Programmer, Upload SPIFFS image, Update platforms and libraries).
• cmd-alt-g/ctrl-alt-g/f4 cycles through causes of build error.
• cmd-alt-h/ctrl-alt-h/shift-f4 goes to the first build error.
• cmd-alt-v/ctrl-alt-v/f8 toggles the build panel.
• escape terminates build / closes the build window.


Intelligent Code Completion

PlatformIO IDE uses clang for the Intelligent Code Completion. To install it or check if it is already installed, please follow see the step II. Clang for Intelligent Code Completion from Installation guide.

Warning: The libraries which are added/installed after the initializing process will not be reflected in the code linter. You need Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter).

Smart Code Linter

PlatformIO IDE uses PlatformIO’s pre-built GCC toolchains for Smart Code Linter and rapid professional development. The configuration data are located in .gcc-flags.json. This file will be automatically created and preconfigured when you initialize project using Menu: PlatformIO > Initialize new PlatformIO Project or update existing….

Warning: If some libraries are not visible in PlatformIO IDE for Atom and Code Completion or Code Linting does not work properly, please perform Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter).
Install Shell Commands

Please navigate to PlatformIO Core Install Shell Commands.

Known issues

Smart Code Linter is disabled for Arduino files

*Smart Code Linter* is disabled by default for Arduino files (*.ino and .pde) because they are not valid C/C++ based source files:

1. Missing includes such as `#include <Arduino.h>`
2. Function declarations are omitted.

There are two solutions:

- Convert Arduino file to C++ manually
- Force Arduino file as C++

Convert Arduino file to C++ manually

Recommended! See *Convert Arduino file to C++ manually*.

Force Arduino file as C++

To force Smart Code Linter to use Arduino files as C++ please

1. Open `.gcc-flags.json` file from the Initialized/Imported project and add `-x c++` flag at the beginning of the value of `gccDefaultCppFlags` field:

```json
{
    "execPath": "...",
    "gccDefaultCFlags": "...",
    "gccDefaultCppFlags": "-x c++ -fsyntax-only ...",
    "gccErrorLimit": 15,
    "gccIncludePaths": "...",
    "gccSuppressWarnings": false
}
```

2. Perform all steps from *Convert Arduino file to C++ manually* (without renaming to .cpp).

**Warning:** Please do not modify other flags here. They will be removed on a next “Project Rebuild C/C++ Index” stage. Please use `build_flags` for “platformio.ini” (Project Configuration File) instead.

Arch Linux: PlatformIO IDE Terminal issue

Please read this article Installing PlatformIO on Arch Linux.
Frequently Asked Questions

Keep build panel visible

PlatformIO IDE hides build panel on success by default. Nevertheless, you can keep it visible all time. Please follow to Menu: PlatformIO > Settings > Build and set Panel Visibility to Keep Visible.

Key-bindings (toggle panel):
- cmd+alt+v - Mac OS X
- ctrl+alt+v - Windows/Linux

Automatically save on build

If you want automatically save all edited files when triggering a build, please follow to Menu: PlatformIO > Settings > Build and check Automatically save on build.

Jump to Declaration

Click on a function/include, press F3 and you will be taken directly to the declaration for that function.

Code Formatting

You need to install atom-beautify package and C/C++ Uncrustify Code Beautifier.

Uninstall Atom with PlatformIO IDE

Here’s how to uninstall the PlatformIO IDE for multiple OS.

See Uninstall PlatformIO Core and dependent packages, if you do not need it in a system.

Windows

1. Uninstall Atom using “Start > Control Panel > Programs and Features > Uninstall”
2. Remove C:\Users\<user name>\.atom folder (settings, packages, etc...)
3. Remove C:\Users\<user name>\AppData\Local\atom folder (application itself)
4. Remove C:\Users\<user name>\AppData\Roaming\Atom folder (cache, etc.)
5. Remove registry records using regedit:
   - HKEY_CLASSES_ROOT\Directory\Background\shell
   - HKEY_CLASSES_ROOT\Directory\shell
   - HKEY_CLASSES_ROOT*\shell
macOS

Run these commands in system Terminal

```bash
rm -rf ~/.atom
rm /usr/local/bin/atom
rm /usr/local/bin/apm
rm -rf /Applications/Atom.app
rm ~/Library/Preferences/com.github.atom.plist
rm ~/Library/Application\ Support/com.github.atom.ShipIt
rm -rf ~/Library/Application\ Support/Atom
rm -rf ~/Library/Saved\ Application\ State/com.github.atom.savedState
rm -rf ~/Library/Caches/com.github.atom
rm -rf ~/Library/Caches/Atom
```

Linux

Run these commands in system Terminal

```bash
rm /usr/local/bin/atom
rm /usr/local/bin/apm
rm -rf ~/atom
rm -rf ~/.atom
rm -rf ~/.config/Atom-Shell
rm -rf /usr/local/share/atom/
```

Articles / Manuals

- Mar, 31, 2017 - Robin Reiter - A little guide to PlatformIO. As an Arduino developer, you may want to check that out! (video review)
- Dec 13, 2016 - Dr. Patrick Mineault - Multi-Arduino projects with PlatformIO
- Nov 10, 2016 - PiGreek - PlatformIO the new Arduino IDE ?!
- Aug 18, 2016 - Primal Cortex - Installing PlatformIO on Arch Linux
- Jul 26, 2016 - Embedded Systems Laboratory - PlatformIO IDE Arduino ESP8266 (Get started with PlatformIO IDE for Arduino board and ESP8266, Thai)
- May 30, 2016 - Ron Moerman - IoT Development with PlatformIO
- May 01, 2016 - Pedro Minatel - PlatformIO – Uma alternativa ao Arduino IDE (PlatformIO - An alternative to the Arduino IDE, Portuguese)
- Apr 23, 2016 - Al Williams - Hackaday: Atomic Arduino (and Other) Development
- Apr 16, 2016 - Sathittham Sangthong - [PlatformIO] PlatformIO Arduino IDE (Let’s play together with PlatformIO IDE [alternative to Arduino IDE], Thai)
- Apr 11, 2016 - Matjaz Trecek - Top 5 Arduino integrated development environments
- Apr 06, 2016 - Aleks - PlatformIO ausprobiert (Tried PlatformIO, German)
- Apr 02, 2016 - Diego Pinto - Você tem coragem de abandonar a IDE do Arduino? PlatformIO + Atom (Do you dare to leave the Arduino IDE? PlatformIO + Atom, Portuguese)
- Mar 30, 2016 - Brandon Cannaday - Getting Started with PlatformIO and ESP8266 NodeMcu
Changelog

Please visit releases page.

CLion

CLion is a cross-platform C/C++ IDE for Linux, OS X, and Windows. CLion includes such features as a smart editor, code generation, code quality assurance, automated refactorings, on-the-fly code analysis, project manager, integrated version control systems and debugger.

Refer to the CLion Documentation page for more detailed information.
Contents

- Installation
- Known issues
  - Project configuration and CMake
  - CLion does not load project build environments from "platformio.ini"
  - Arduino .ino files are not supported
• **Quick Start**
  – *Setting Up the Project*
  – *Build & Upload*
    * Configuration options
• **Device/Serial Monitor**
• **Debugging**
  – *Peripheral Registers*
• **PlatformIO Home**

## Installation

1. Install PlatformIO Core (CLI). See *Installation* guide

   **Warning:** This a temporary step and depends on CLion #CPP-19412: Automatically install PlatformIO Core

2. Download and install CLion IDE

3. Install official PlatformIO for CLion plugin. Open “Configure > Plugins” window and go to the “Marketplace” tab. Search for PlatformIO and press the “Install” button.
1. Open "Configure > Plugins > Marketplace"

2. Search for "PlatformIO"

3. Install plugin
4. Restart CLion IDE.

Note: We also recommend to install Ini plugin that provides syntax highlighting, formatting, code folding, and viewing structure for "platformio.ini" (Project Configuration File).

Known issues

Project configuration and CMake

PlatformIO does not depend on “CMake” and uses own multi-platform build system. Project configuration, such as build flags, library dependencies, etc., should be declared in “platformio.ini” (Project Configuration File).

Warning: Please note that “PlatformIO for CLion” plugin does not update automatically CLion configuration. There is a feature request CLion #CPP-18367: Follow platformio.ini changes and update the project.

Update CLion configuration: “Tools > PlatformIO > Re-Init”

CLion does not load project build environments from “platformio.ini”

This is a known issue CLion CPP-19478: CLion does not handle “CMAKE_CONFIGURATION_TYPES” from CMakeLists.txt. A temporary solution is to manually configure project profiles in CLion.

Please open CLion Settings and navigate to “Build, Execution, Deployment > CMake”. Press “+” button and PlatformIO-based project profiles:
Arduino `.ino` files are not supported

CLion uses “CMake” tool for code completion and code linting. As a result, it doesn’t support the Arduino files (*.ino and .pde) because they are not valid C/C++ based source files:

1. Missing includes such as #include `<Arduino.h>`
2. Function declarations are omitted.

See how to [Convert Arduino file to C++ manually](#).

**Quick Start**

This tutorial introduces you to the basics of PlatformIO for CLion workflow and shows you a creation process of a simple “Blink” example. After finishing you will have a general understanding of how to work with projects in the CLion IDE.
Setting Up the Project

1. Please open “New Project” wizard, select board and framework, and create a new PlatformIO project. Please **USE ONLY** Latin characters (a-z) in a project name to avoid further issues with project compilation. Numbers and some symbols are allowed depending on a file system:

   ![PlatformIO New Project Wizard](image)

   **Project Name**
   - only [a-z0-9_] characters

2. Rename `main.c` file in `src` folder to `main.cpp` (right click on a file `main.c`, Refactor > Rename…).

   Paste the next contents:

   ```
   Warning: The code below works only in pair with Arduino-based boards. Please follow to PlatformIO Project Examples repository for other pre-configured projects.
   
   /*! Blink
   *
   * Turns on an LED on for one second,
   * then off for one second, repeatedly.
   */
   
   #include "Arduino.h"
   // Set LED_BUILTIN if it is not defined by Arduino framework
   // #define LED_BUILTIN 13

   void setup()
   {
     // initialize LED digital pin as an output.
     pinMode(LED_BUILTIN, OUTPUT);
   }
   ```

   (continues on next page)
void loop()
{
    // turn the LED on (HIGH is the voltage level)
    digitalWrite(LED_BUILTIN, HIGH);

    // wait for a second
    delay(1000);

    // turn the LED off by making the voltage LOW
    digitalWrite(LED_BUILTIN, LOW);

    // wait for a second
    delay(1000);
}
Build & Upload

1. Open project configuration wizard and add PlatformIO Upload and PlatformIO Debug configurations (you can add the rest configurations if you need them):

2. Remove pre-task (“Build”) from PlatformIO Upload and PlatformIO Debug configurations to avoid double project building. You should see “There are no tasks to run before launch”.

2662 Chapter 1. Contents
3. Select PlatformIO Upload configuration and use the “Build” button for project compilation or the “Run” for a firmware uploading:
Configuration options

- **Build options**
- **Upload options**

Device/Serial Monitor

Please open CLion terminal and use *Device Manager CLI*. You can also configure device monitor per project using *Monitor options*.

Please note that you need to manually close/stop device monitor before firmware uploading.
PlatformIO Documentation, Release 5.0.5a1

Debugging

“PlatformIO for CLion” supports Debugging which allows you to debug your application with 1-Click without extra steps. Please select PlatformIO Debug configuration and press the “Debug” button:

Peripheral Registers

Please navigate to the “Peripheral” tab in a “Debug” view, press the “Configure” icon and select registers to monitor. Close configuration window.

Warning: Currently, CLion does not load automatically Peripheral Register Definitions (SVD file) provided by PlatformIO. There is a feature request CLion #CPP-18369: Support CLION_SVD_FILE_PATH CMake variable
to auto-find svd file
1.19. Cloud & Desktop IDE
PlatformIO Home


Nevertheless, you can start it manually and open in your browser. Please note that some features do not work, such as project opening. Please open CLion terminal and run `pio home` command:

```
pio -c clion home
```

Further for reading:

- [Tutorials and Examples](#) (step-by-step tutorials with debugging and unit testing)
- [CLion documentation](#)

**Happy coding with PlatformIO!**

**CodeBlocks**

Code::Blocks is a free, open-source cross-platform IDE that supports multiple compilers including GCC, Clang and Visual C++. It is developed in C++ using wxWidgets as the GUI toolkit. Using a plugin architecture, its capabilities and features are defined by the provided plugins. Currently, Code::Blocks is oriented towards C, C++, and Fortran.

CodeBlocks IDE can be downloaded from [here](#).
Contents

- CodeBlocks
  - Integration

Integration

Integration process consists of these steps:

1. Open system Terminal and install PlatformIO Core (CLI)
2. Create new folder for your project and change directory (`cd`) to it
3. Generate a project using PlatformIO Core Project Generator (`pio project init --ide`)
4. Import project in IDE.

Choose board ID using `pio boards` or Embedded Boards Explorer command and generate project via `pio project init --ide` command:

```
PIO project init --ide codeblocks --board <ID>
```

(continues on next page)
For example, generate project for Arduino UNO
pio project init --ide codeblocks --board uno

Then:

1. Open this project via Menu: File > Open...
2. Add new files to src directory (*.c, *.cpp, *.ino, etc.) via Menu: File > New > File..
3. Build project using Menu: Build > Build
4. Upload firmware using Menu: Build > Run

**Warning:** The libraries which are added, installed or used in the project after generating process won’t be reflected in IDE. To fix it you need to reinstall project using `pio project init` (repeat it).

**Eclipse**

The **Eclipse CDT (C/C++ Development Tooling)** Project provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform. Features include: support for project creation and managed build for various toolchains, standard make build, source navigation, various source knowledge tools, such as type hierarchy, call graph, include browser, macro definition browser, code editor with syntax highlighting, folding and hyperlink navigation, source code refactoring and code generation, visual debugging tools, including memory, registers, and disassembly viewers.

Refer to the [CDT Documentation](#) page for more detailed information.
Contents

- Integration
- Live Integration
- Debugging
- Articles / Manuals

1.19. Cloud & Desktop IDE
Integration

Integration process consists of these steps:

1. Open system Terminal and install PlatformIO Core (CLI)
2. Create new folder for your project and change directory (cd) to it
3. Generate a project using PlatformIO Core Project Generator (pio project init --ide)
4. Import project in IDE.

Choose board ID using pio boards or Embedded Boards Explorer command and generate project via pio project init --ide command:

```
pio project init --ide eclipse --board <ID>
# For example, generate project for Arduino UNO
pio project init --ide eclipse --board uno
```

Then:

1. Import this project via Menu: File > Import... > General > Existing Projects into Workspace > Next and specify root directory where is located “platformio.ini” (Project Configuration File)
2. Open source file from src directory (*.c, *.cpp, *.ino, etc.)
3. Build project using Menu: Project > Build Project or pre-configured Make Targets (see screenshot below):
   - PlatformIO: Build - Build project without auto-uploading
   - PlatformIO: Clean - Clean compiled objects.
   - PlatformIO: Test - Unit Testing
   - PlatformIO: Upload - Build and upload (if no errors)
   - PlatformIO: Upload using Programmer see Upload using Programmer
   - PlatformIO: Upload SPIFFS image see Using Filesystem
   - PlatformIO: Update platforms and libraries - Update installed platforms and libraries via pio update
   - PlatformIO: Rebuild C/C++ Project Index - Rebuild C/C++ Index for the Project. Allows one to fix code completion and code linting issues.

If you have some problems with unresolved includes, defines, etc., then

1. Rebuild PlatformIO Project Index: Menu: PlatformIO: Rebuild C/C++ Project Index target
2. Rebuild Eclipse Project Index: Menu: Project > C/C++ Index > Rebuild
3. Refresh Project, right click on the project Project > Refresh (F5) or restart Eclipse IDE.

**Warning:** The libraries which are added, installed or used in the project after generating process won’t be reflected in IDE. To fix it please run PlatformIO: Rebuild C/C++ Project Index target and right click on the project and Project > Refresh (F5).
Warning: The C/C++ GCC Cross Compiler Support package must be installed in Eclipse, otherwise the CDT Cross GCC Built-in Compiler Settings provider will not be available (check the Providers tab in Project > Properties > C/C++ General > Preprocessor Include Paths, Macros etc. for a marked entry named CDT Cross GCC Built-in Compiler Settings).

If this provider is not available, toolchain related includes cannot be resolved.

Live Integration

Eclipse Virtual IoT Meetup: PlatformIO: a cross-platform IoT solution to build them all!

Debugging

A debugging feature is provided by Debugging and new debug configuration named “PlatformIO Debugger” is created. No need to do extra configuration steps:

1. Build a project first time or after “Clean” operation using PlatformIO: Build target
2. Launch debugger via “Menu: Debug” or “Bug Icon” button on Tool Bar.
3. Wait for a while, PlatformIO will prepare project for debugging and session will be started soon.

Articles / Manuals

- May 05, 2016 - Ivan Kravets, Ph.D. / Eclipse Virtual IoT Meetup - PlatformIO: a cross-platform IoT solution to build them all!
Emacs

GNU Emacs is an extensible, customizable text editor - and more. At its core is an interpreter for Emacs Lisp, a dialect of the Lisp programming language with extensions to support text editing.

Refer to the Emacs Documentation page for more detailed information.

Contents

- Emacs
  - Integration
    * Project Generator
    * PlatformIO-Mode
  * Code Completion and Navigation
Integration

Integration process consists of these steps:

1. Open system Terminal and install PlatformIO Core (CLI)
2. Create new folder for your project and change directory (cd) to it
3. Generate a project using PlatformIO Core Project Generator (pio project init --ide)
4. Open project in Emacs.

Project Generator

Choose board ID using pio boards or Embedded Boards Explorer command and generate project via pio project init --ide command:

```
pio project init --ide emacs --board <ID>
```

**Warning:** The libraries which are added, installed or used in the project after generating process won’t be reflected in IDE. To fix it you need to reinitialize project using pio project init (repeat it).

PlatformIO-Mode

An Emacs minor mode has been written to facilitate building and uploading from within Emacs. It can be installed from the MELPA repository using M-x package-install. See the MELPA Getting Started page for more information.

Setup instructions and an example config can be found at the Github page.

There are 6 predefined targets for building.

- `platformio_build` - Build project without auto-uploading. (C-c i b)
- `platformio_clean` - Clean compiled objects. (C-c i c)
- `platformio_upload` - Build and upload (if no errors). (C-c i u)
- `platformio_programmer_upload` - Build and upload using external programmer (if no errors, see Upload using Programmer). (C-c i p)
- `platformio_spiffs_upload` - Upload files to file system SPIFFS (see Using Filesystem). (C-c i s)
- `platformio_update` - Update installed platforms and libraries. (C-c i d)

Code Completion and Navigation

Please install the next:

- C/C++/ObjC language server supporting cross references, hierarchies, completion and semantic highlighting
- Emacs client for ccls, a C/C++ language server.
NetBeans

NetBeans is a Java-based integrated development environment (IDE). It provides out-of-the-box code analyzers and editors for working with the latest Java 8 technologies—Java SE 8, Java SE Embedded 8, and Java ME Embedded 8. The IDE also has a range of new tools for HTML5/JavaScript, in particular for Node.js, KnockoutJS, and AngularJS; enhancements that further improve its support for Maven and Java EE with PrimeFaces; and improvements to PHP and C/C++ support.

NetBeans IDE can be downloaded from here. Just make sure you download the C/C++ version (or if you already use NetBeans, install the C/C++ development plugins).

**Warning:** Note that the version of NetBeans IDE provided by some Linux package repositories may be a Flatpak version, which are known to have issues with accessing the PlatformIO CLI installed due to the container nature of this technology. It is recommended that you use the native version of the NetBeans IDE via the above link.
Contents

- NetBeans
  - Integration
  - Articles / Manuals

1.19. Cloud & Desktop IDE
Integration

Integration process consists of these steps:

1. Open system Terminal and install *PlatformIO Core (CLI)*
2. Create new folder for your project and change directory (`cd`) to it
3. Generate a project using PlatformIO Core Project Generator (`pio project init --ide`)
4. Import project in IDE.

Choose board ID using `pio boards` or Embedded Boards Explorer command and generate project via `pio project init --ide` command:

```
pio project init --ide netbeans --board <ID>
# For example, generate project for Arduino UNO
pio project init --ide netbeans --board uno
```

Then:

1. Open this project via Menu: File > Open Project...
2. Add new files to `src` directory (*.c, *.cpp, *.ino, etc.) via right-click on `src` folder in the “Projects” pane
3. Build project using Menu: Run > Build Project
4. Upload firmware using Menu: Run > Run Project

**Warning:** The libraries which are added, installed or used in the project after generating process won’t be reflected in IDE. To fix it you need to reinitialize project using `pio project init` (repeat it).

Articles / Manuals

- Feb 22, 2016 - Grzegorz Holdys - How to Integrate PlatformIO with Netbeans

See the full list with *Articles about us*.

Qt Creator

The *Qt Creator* is an open source cross-platform integrated development environment. The editor includes such features as syntax highlighting for various languages, project manager, integrated version control systems, rapid code navigation tools and code autocompletion.

Refer to the *Qt-creator Manual* page for more detailed information.
Integration process consists of these steps:

1. Open system Terminal and install *PlatformIO Core (CLI)*
2. Create new folder for your project and change directory (`cd`) to it
3. Generate a project using PlatformIO Core Project Generator (`pio project init --ide`)
4. Import project in IDE.
Project Generator

Choose board ID using `pio boards` or Embedded Boards Explorer command and generate project via `pio project init --ide` command:

```
pio project init --ide qtcreator --board <ID>
```

# For example, generate project for Arduino UNO
```
pio project init --ide qtcreator --board uno
```

Then:

1. Import project via File > Open File or Project and select `platformio.pro` from the folder where is located "platformio.ini" (Project Configuration File)

2. Select default desktop kit and click on Configure Project (Projects mode, left panel)

3. Set General > Build directory to the project directory where is located "platformio.ini" (Project Configuration File)

4. Remove all items from Build Steps, click on Build Steps > Add Build Step > Custom Process Step and set:
   - **Command**: platformio
   - **Arguments**: `-f -c qtcreator run
   - **Working directory**: `%{buildDir}`

5. Remove all items from Clean Steps, click on Clean Steps > Add Clean Step > Custom Process Step and set:
   - **Command**: platformio
   - **Arguments**: `-f -c qtcreator run --target clean
   - **Working directory**: `%{buildDir}`

6. Update PATH in Build Environment > PATH > EDIT with the result of this command (paste in Terminal):

   ```
   # Linux, Mac
   echo $PATH
   # Windows
   echo %PATH%
   ```

7. Switch to Edit mode (left panel) and open source file from src directory (*.c, *.cpp, *.ino, etc.)

8. Build project: Menu: Build > Build All.
Warning: The libraries which are added, installed or used in the project after generating process won’t be reflected in IDE. To fix it you need to reinitialize project using `pio project init` (repeat it).

Manual Integration

Setup New Project

First of all, let’s create new project from Qt Creator Start Page: New Project or using Menu: File > New File or Project, then select project with Empty Qt Project type (Other Project > Empty Qt...
Project), fill Name, Create in.

On the next steps select any available kit and click Finish button.
Secondly, we need to delete default build and clean steps and configure project with PlatformIO Build System (click on Projects label on left menu or Ctrl+5 shortcut):
Thirdly, change project file by adding path to directories with header files. Please edit project file to match the following contents:

```
win32 {
    HOMEDIR += $$\$(USERPROFILE)
}
else {
    HOMEDIR += $$\$(HOME)
}

INCLUDEPATH += "$$\$(HOMEDIR)/.platformio/packages/framework-arduinoavr/cores/arduino"
INCLUDEPATH += "$$\$(HOMEDIR)/.platformio/packages/toolchain-atmelavr/avr/include"
```
First program in Qt Creator

Simple “Blink” project will consist from two files: 1. In the console, navigate to the root of your project folder and initialize pio project with `pio project init` 2. The main “C” source file named `main.c` must be located in the `src` directory. Let’s create new text file named `main.c` using Menu: New File or Project > General > Text File:
Copy the source code which is described below to file `main.c`.

```c
#include "Arduino.h"
#define WLED 13  // Most Arduino boards already have an LED attached to pin 13 on the board itself

void setup()
{
    pinMode(WLED, OUTPUT);  // set pin as output
}

void loop()
{
    digitalWrite(WLED, HIGH);  // set the LED on
delay(1000);                  // wait for a second
digitalWrite(WLED, LOW);      // set the LED off
delay(1000);                  // wait for a second
}
```

3. Locate the project configuration file named `platformio.ini` at the root of the project directory and open it.
Edit the content to match the code described below.

```plaintext
; PlatformIO Project Configuration File
;
; Build options: build flags, source filter, extra scripting
; Upload options: custom port, speed and extra flags
; Library options: dependencies, extra library storages
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:arduino_uno]
platform = atmelavr
framework = arduino
board = uno
```

Conclusion

Taking everything into account, we can build project with shortcut Ctrl+Shift+B or using Menu: Build > Build All.
Sublime Text

The Sublime Text is a cross-platform text and source code editor, with a Python application programming interface (API). Sublime Text is proprietary software. Its functionality is extendable with plugins. Most of the extending packages have free-software licenses and are community-built and maintained. Sublime Text lacks graphical setting dialogues and is entirely configured by editing text files.

Refer to the Sublime Text Documentation page for more detailed information.

Contents

- Deviot Plugin
- Integration
  - Project Generator
  - Manual Integration
    - Initial configuration
    - Command Hotkeys
Deviot Plugin

We are glad to inform you about an awesome Sublime Text plugin for IoT development named Deviot. It is based on PlatformIO Core (CLI) and will automatically install it for you. Please visit official Deviot page for the further installation steps and documentation.

Integration

Project Generator

Integration process consists of these steps:

1. Open system Terminal and install **PlatformIO Core (CLI)**
2. Create new folder for your project and change directory (cd) to it
3. Generate a project using PlatformIO Core Project Generator (**pio project init --ide**) 
4. Import project in IDE.

Choose board ID using **pio boards** or **Embedded Boards Explorer** command and generate project via **pio project init --ide** command:

```
pio project init --ide sublimetext --board <ID>
```

# For example, generate project for Arduino UNO
```
pio project init --ide sublimetext --board uno
```

Then:

1. Import project via Menu: Project > Open Project... and select platformio.
   sublimetext-project from the folder where is located “platformio.ini” (Project Configuration File)
2. Select PlatformIO as build system: Menu: Tools > Build System > PlatformIO
3. Open source file from src directory (*.c, *.cpp, *.ino, etc.)

Also, you can access to all pre-configured targets via Menu: Tools > Builds With... (ST3)

- **PlatformIO - Build** - Build project without auto-uploading
- **PlatformIO - Clean** - Clean compiled objects.
- **PlatformIO - Test** - **Unit Testing**
- **PlatformIO - Upload** - Build and upload (if no errors)
- **PlatformIO - Upload using Programmer** see Upload using Programmer

1.19. Cloud & Desktop IDE 2689
- PlatformIO - Upload SPIFFS image see *Using Filesystem*
- PlatformIO - Update platforms and libraries - Update installed platforms and libraries via *pio update*.

**Manual Integration**

**Note:** Please verify that folder where is located `platformio` program is added to PATH (wiki) environment variable.

**Initial configuration**

First of all, we need to create “New Build System” with name “PlatformIO” from Menu: **Tools > Build System > New Build System** and fill it like described below:

```json
{
    "cmd": ["platformio", "-f", "--c", "sublimetext", "run"],
    "working_dir": "${project_path:${folder}}",
    "variants": [
        {
            "name": "Clean",
            "cmd": ["platformio", "-f", "--c", "sublimetext", "run", "--target", "clean"]
        },
        {
            "name": "Upload",
            "cmd": ["platformio", "-f", "--c", "sublimetext", "run", "--target", "upload"]
        }
    ]
}
```

Secondly, we need to select “PlatformIO” Build System from a list:
After that, we can use the necessary commands from **Menu: Tools > Command Palette** or with **Ctrl+Shift+P** (Windows/Linux) **Cmd+Shift+P** (Mac) shortcut.
Command Hotkeys

Sublime Text allows one to bind own hotkey per command. Let’s setup them for PlatformIO commands using shortcut

Menu:  Preferences > Key-Bindings - User:
We are going to use these shortcuts:

- F11 for clean project
- F12 for upload firmware to target device

In this case, the final code will look like:

```
[{
  "keys": ["f11"],
  "command": "build",
  "args": {
    "variant": "Clean"
  }
},
{
  "keys": ["f12"],
  "command": "build",
  "args": {
    "variant": "Upload"
  }
}]
```

**First program in Sublime Text**

Simple “Blink” project will consist from two files:

1. Main “C” source file named `main.c` must be located in the `src` directory. Let’s create new file named `main.c` using Menu: File > New File or shortcut Ctrl+N (Windows/Linux) Cmd+N (Mac) with the next contents:

```c
#include "Arduino.h"
#define WLED 13 // Most Arduino boards already have an LED attached to pin 13 on the board itself

void setup()
{
  pinMode(WLED, OUTPUT); // set pin as output
}
```

(continues on next page)
void loop()
{
    digitalWrite(WLED, HIGH); // set the LED on
    delay(1000); // wait for a second
    digitalWrite(WLED, LOW); // set the LED off
    delay(1000); // wait for a second
}

2. Project Configuration File named platformio.ini must be located in the project root directory. Copy the source code which is described below to it.

```ini
; PlatformIO Project Configuration File
; Build options: build flags, source filter, extra scripting
; Upload options: custom port, speed and extra flags
; Library options: dependencies, extra library storages
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:arduino_uno]
platform = atmelavr
framework = arduino
board = uno
```

Conclusion

Taking everything into account, we can open project directory in Sublime Text using Menu: File > Open Folder and build it with shortcut Ctrl+B (Windows/Linux) or Cmd+B (Mac), clean project with shortcut F11 and upload firmware to target with shortcut F12.

Debugging

A debugging feature is provided by Debugging and new debug configuration named “PlatformIO Debugger“ is created. No need to do extra configuration steps!

1. Install SublimeGDB package
2. Launch debugger with F5
3. Wait for a while, PlatformIO will prepare project for debugging and session will be started soon.

Vim

Vim is an open-source, powerful and configurable text editor. Vim is designed for use both from a command-line interface and as a standalone application in a graphical user interface.
Integration

Integration process consists of these steps:

1. Open system Terminal and install PlatformIO Core (CLI)
2. Create new folder for your project and change directory (cd) to it
3. Generate a project using PlatformIO Core Project Generator (pio project init --ide)
4. Import project in IDE.
Project Generator

Choose board ID using `pio boards` or Embedded Boards Explorer command and generate project via `pio project init --ide` command:

```
pio project init --ide vim --board <ID>
```

Recommended bundles:

- C/C++/ObjC language server supporting cross references, hierarchies, completion and semantic highlighting
- Async language server protocol plugin for vim and neovim
- “Neomake-PlatformIO” Plugin

Put to the project directory Makefile wrapper with contents:

```
# Uncomment lines below if you have problems with $PATH
#SHELL := /bin/bash
#PATH := /usr/local/bin:$(PATH)

all:
    pio -f -c vim run

upload:
    pio -f -c vim run --target upload

clean:
    pio -f -c vim run --target clean

program:
    pio -f -c vim run --target program

uploadfs:
    pio -f -c vim run --target uploadfs

update:
    pio -f -c vim update
```

Pre-defined targets:

- **Build**: Build project without auto-uploading
- **Clean**: Clean compiled objects.
- **Upload**: Build and upload (if no errors)
- **Upload using Programmer** see *Upload using Programmer*
- **Upload SPIFFS image** see *Using Filesystem*
- **Update platforms and libraries** - Update installed platforms and libraries via `pio update`.

Now, in VIM `cd /path/to/this/project` and press Ctrl+B or Cmd+B (Mac). PlatformIO should compile your source code from the src directory, make firmware and upload it.

**Note**: If hotkey doesn’t work for you, try to add this line `nnoremap <C-b> :make<CR> to ~/.vimrc`
Articles / Manuals

- Arduino : vim + platformio (Arduino development at the command line: VIM + PlatformIO, Japanese)

See a full list with Articles about us.

Visual Studio

The Microsoft Visual Studio (Free) is an integrated development environment (IDE) from Microsoft. Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring.

Refer to the Visual Studio Documentation page for more detailed information.

Contents

- Visual Studio
  - Integration
    * Project Generator
    * Manual Integration
  - Setup New Project

1.19. Cloud & Desktop IDE 2697
Integration

Integration process consists of these steps:

1. Open system Terminal and install PlatformIO Core (CLI)
2. Create new folder for your project and change directory (cd) to it
3. Generate a project using PlatformIO Core Project Generator (pio project init --ide)
4. Import project in IDE.

Project Generator

Choose board ID using pio boards or Embedded Boards Explorer command and generate project via pio project init --ide command:

```bash
pio project init --ide visualstudio --board <ID>
```

# For example, generate project for Arduino UNO
pio project init --ide visualstudio --board uno

Then:

1. Import this project via Menu: File > Open > Project/Solution and specify root directory where is located “platformio.ini” (Project Configuration File)
2. Open source file from src directory (*.c, *.cpp, *.ino, etc.)

**Warning:** The libraries which are added, installed or used in the project after generating process won’t be reflected in IDE. To fix it you need to reinitialize project using pio project init (repeat it).

Manual Integration

Setup New Project

First of all, let’s create new project from Visual Studio Start Page: Start > New Project or using Menu: File > New > Project, then select project with Makefile type (Visual C++ > General > Makefile Project), fill Project name, Solution name, Location fields and press OK button.
Secondly, we need to configure project with PlatformIO Build System:
If we want to use native AVR programming, we have to specify additional preprocessor symbol (“Preprocessor definitions” field) about your MCU. For example, an Arduino Uno is based on the ATmega328 MCU. In this case we will add new definition `__AVR_ATmega328__`. 
Release Configuration is the same as Debug, so on the next step we check “Same as Debug Configuration” and click “Finish” button.
Thirdly, we need to add directories with header files using project properties (right click on the project name or Alt-Enter shortcut) and add two directories to Configuration Properties > NMake > Include Search Path:

```
$(HOMEDRIVE)$HOME\platformio\packages\toolchain-atmelavr\avr\include
$(HOMEDRIVE)$HOME\platformio\packages\framework-arduinoavr\cores\arduino
```
First program in Visual Studio

Simple “Blink” project will consist from two files:

1. Main “C++” source file named `main.cpp` must be located in the `src` directory. Let’s create new file named `main.cpp` using Menu: File > New File or shortcut Ctrl+N:
Copy the source code which is described below to file `main.cpp`.

```c
#include "Arduino.h"

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT); // set pin as output
}

void loop()
{
    digitalWrite(LED_BUILTIN, HIGH); // set the LED on
    delay(1000); // wait for a second
    digitalWrite(LED_BUILTIN, LOW); // set the LED off
    delay(1000); // wait for a second
}
```

2. Project Configuration File named `platformio.ini` must be located in the project root directory.
Copy the source code which is described below to it.

```plaintext
; PlatformIO Project Configuration File
;
; Build options: build flags, source filter, extra scripting
; Upload options: custom port, speed and extra flags
; Library options: dependencies, extra library storages
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:arduino_uno]
platform = atmelavr
framework = arduino
board = uno
```

**Conclusion**

Taking everything into account, we can build project with shortcut Ctrl+Shift+B or using Menu: Build > Build Solution.

**Known issues**
IntelliSense Errors

VS Studio does not allow one to specify for project other toolchain which will be used by IntelliSense. In this case, IntelliSense does not understand GCC-specific definitions.

However, these errors does not have any influence on PlatformIO Build System. It means that you can ignore them and rely on PlatformIO Build System messages which will be shown in output console after build.

Nevertheless, you can provide an IntelliSense-friendly definition of problematic GCC constructs and make sure that the GCC will ignore such definitions or disable IntelliSense error reporting at all. See details in issue #543

VSCode

PlatformIO IDE is the next-generation integrated development environment for IoT.

- Cross-platform build system without external dependencies to the OS software:
  - 800+ Boards
  - 35+ Development Platforms
  - 20+ Frameworks
- Debugging
• Remote Development

• Unit Testing

• C/C++ Intelligent Code Completion

• C/C++ Smart Code Linter for rapid professional development

• Library Manager for the hundreds popular libraries

• Multi-projects workflow with multiple panes

• Themes support with dark and light colors

• Serial Port Monitor

• Built-in Terminal with PlatformIO Core (CLI) and CLI tool (pio, platformio)

• Built-in PlatformIO Home.

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Python, PHP, Go) and runtimes (such as .NET and Unity)
Contents

- Installation
- Quick Start
  - Setting Up the Project
- PlatformIO Toolbar
- PlatformIO Core (CLI)
- Project Tasks
  - Task Explorer
  - Task Runner
  - Custom Tasks
• Multi-project Workspaces
• Serial Port Monitor
• Debugging
  – Variable Format
  – Watchpoints
• Install Shell Commands
• Proxy Server Support
• Key Bindings
• Settings
  – platformio-ide.activateOnlyOnPlatformIOProject
  – platformio-ide.autoCloseSerialMonitor
  – platformio-ide.autoRebuildAutocompleteIndex
  – platformio-ide.buildTask
  – platformio-ide.autoPreloadEnvTasks
  – platformio-ide.customPATH
  – platformio-ide.disableToolbar
  – platformio-ide.forceUploadAndMonitor
  – platformio-ide.reopenSerialMonitorDelay
  – platformio-ide.useBuiltinPython
  – platformio-ide.useBuiltinPIOCore
  – platformio-ide.useDevelopmentPIOCore
  – platformio-ide.disablePIOHomeStartup
  – platformio-ide.pioHomeServerHttpPort
• Known issues
  – PackageManager is unable to install tool
• Changelog

Installation

Note: Please note that you do not need to install PlatformIO Core (CLI) separately if you are going to use VSCode. PlatformIO Core (CLI) is built into PlatformIO IDE and you will be able to use it within PlatformIO IDE Terminal.

0. Download and install official Microsoft Visual Studio Code. PlatformIO IDE is built on top of it
1. Open VSCode Package Manager
2. Search for the official platformio ide extension
3. Install PlatformIO IDE.
Quick Start

This tutorial introduces you to the basics of PlatformIO IDE workflow and shows you a creation process of a simple “Blink” example. After finishing you will have a general understanding of how to work with projects in the IDE.

Setting Up the Project

1. Click on “PlatformIO Home” button on the bottom PlatformIO Toolbar
2. Click on “New Project”, select a board and create new PlatformIO Project
3. Open `main.cpp` file form `src` folder and replace its contents with the next:

```cpp
Warning: The code below works only in pair with Arduino-based boards. Please follow to PlatformIO Project Examples repository for other pre-configured projects.

```cpp
/**
* Blink
*
* Turns on an LED on for one second,
* then off for one second, repeatedly.
*/
#include "Arduino.h"

// Set LED_BUILTIN if it is not defined by Arduino framework
// #define LED_BUILTIN 13

void setup()
{
    // initialize LED digital pin as an output.
    pinMode(LED_BUILTIN, OUTPUT);
} 
```
void loop()
{
    // turn the LED on (HIGH is the voltage level)
    digitalWrite(LED_BUILTIN, HIGH);

    // wait for a second
    delay(1000);

    // turn the LED off by making the voltage LOW
    digitalWrite(LED_BUILTIN, LOW);

    // wait for a second
    delay(1000);
}

4. Build your project with `ctrl+alt+b` hotkey (see all Key Bindings in “User Guide” section below) or using “Build” button on the \textit{PlatformIO Toolbar}
Further for reading:

- *Tutorials and Examples* (step-by-step tutorials with debugging and unit testing)
- Learn more about *PlatformIO Toolbar* and other commands (Upload, Clean, Serial Monitor) below.

**Happy coding with PlatformIO!**

**PlatformIO Toolbar**

PlatformIO IDE Toolbar is located in VSCode Status Bar (left corner) and contains quick access buttons for the popular commands. Each button contains hint (delay mouse on it).

1. *PlatformIO Home*
2. PlatformIO: Build
3. PlatformIO: Upload
4. PlatformIO: Clean
5. **Serial Port Monitor**

6. **PlatformIO Core (CLI)**

7. Project environment switcher (if more than one environment is available). See *Section [env] of “platformio.ini” (Project Configuration File)*.

**PlatformIO Core (CLI)**

There are 2 ways how to access **PlatformIO Core (CLI)**:

1. “Terminal” icon on the **PlatformIO Toolbar**

2. Left Activity Bar > PlatformIO (ant icon) > Quick Access > Miscellaneous > PlatformIO Core CLI

---

**Project Tasks**

**Task Explorer**

PlatformIO provides access to “Project Task” where you can control the build process of the environments declared in “platformio.ini” (*Project Configuration File*). Project Task Explorer is located in the VSCode Activity Bar under the branded PlatformIO icon. You can also access it via “VSCode Menu > Open View... > PlatformIO”.

**Hint:** Please note that you can drag/move “Project Task” into the another view within VSCode, such as “Explorer”.

---

2715
Task Runner

PlatformIO IDE provides built-in tasks through the menu Terminal > Run Task... (Build, Upload, Clean, Monitor, etc) and custom tasks per "platformio.ini" (Project Configuration File) environment ([env:***]). The default behavior is to use Terminal Panels for presentation, one panel dedicated to each unique task.

The PlatformIO IDE provides its own Problems Matcher named $platformio. You can use it later if you decide to change base task settings.

You can override existing tasks with your own presentation options. For example, let’s configure PlatformIO Task Runner to use a NEW Terminal panel for each “Build” command:

1. The menu item Terminal > Run Task... opens up a list of VSCode tasks for PlatformIO. In the line PlatformIO: Build, press the gear icon on the far right side of the list. This creates or opens the file .vscode/tasks.json with some template code.

2. Replace the template in tasks.json with this code
See more options in the official VSCode documentation.

## Custom Tasks

Custom tasks can be added to `tasks.json` file located in the `.vscode` folder in the root of project. Please read the official documentation [Tasks in VSCode](https://code.visualstudio.com/docs/editor/tasks).

This simple example demonstrates a custom monitor task which echoes input locally. There are a lot of other commands, please read more about [PlatformIO Core (CLI)](https://docs.platformio.org/projects/core-en) and its commands ([CLI Guide](https://docs.platformio.org/projects/core-en/)).

```json
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "command": "platformio",
            "args": [
                "device",
                "monitor",
                "--echo"
            ],
            "problemMatcher": [
                "$platformio"
            ],
            "label": "PlatformIO: Monitor (local echo)"
        }
    ]
}
```

If the `platformio` executable file is not in your system environment “PATH”, you can provide the full path to the binary folder using the `options` field for the task. For example, if the `platformio` binary is located in the home folder “~/.platformio/penv/bin”:

```json
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "command": "platformio",
            "args": [
                "device",
                "monitor",
                "--echo"
            ],
            "problemMatcher": [
                "$platformio"
            ],
            "label": "PlatformIO: Monitor (local echo)"
        }
    ]
}
```

(continues on next page)
Multi-project Workspaces

You can work with multiple project folders in Visual Studio Code with multi-root workspaces. This can be very helpful when you are working on several related projects at the same time. Read more in the documentation Multi-root Workspaces.

Serial Port Monitor

You can customize Serial Port Monitor using Monitor options in “platformio.ini” (Project Configuration File):

- **monitor_port**
- **monitor_speed**
- **monitor_rts**
- **monitor_dtr**
- **monitor_flags**

Example:

```ini
[env:esp32dev]
platform = espressif32
framework = arduino
board = esp32dev

; Custom Serial Monitor port
monitor_port = /dev/ttyUSB1

; Custom Serial Monitor speed (baud rate)
monitor_speed = 115200
```

Debugging

Debugging in VSCode works in combination with Debugging. You should have PlatformIO Account to work with it. VSCode has a separate activity view named “Debug” (accessed by the bug icon on the left toolbar). Debugging extends it with more advanced debugging instruments and features:
• Local, Global, and Static Variable Explorer
• Conditional Breakpoints
• Expressions and Watchpoints
• Generic Registers
• Peripheral Registers
• Memory Viewer
• Disassembly
• Multi-thread support
• A hot restart of an active debugging session.

There are two pre-configured debugging configurations:

**PIO Debug Default configuration.** PlatformIO runs the **Pre-Debug** task and builds the project using **Debug Configuration**. Also, it checks for project changes.

**PIO Debug (skip Pre-Debug)** PlatformIO skips the **Pre-Debug** stage and DOES NOT build or check the project for changes. If you do changes in project source files, they will not be reflected in debug sessions until you switch back to the “PIO Debug” configuration or manually run the “Pre-Debug” task.

This configuration is very useful for quick debug session. It is super fast by skipping several checks, letting you control project changes manually.

**Note:** Please note that **Debugging** will use the first declared build environment in “platformio.ini” (Project Configuration File) if the `default_envs` option is not specified.
Variable Format

Currently, VSCode does not provide an UI or API to change the variable format. See the related VSCode Issue #28025. A temporary solution is to set the default numerical base in which the debugger displays numeric output in the Debug Console. (The Debug Console is visible during active debugging sessions). For example, to show variables in hexadecimal format, copy the code below and paste it into “Debug Console”:

```
set output-radix 16
```

Possible values, listed in decimal base, are: 8, 10, 16.

Watchpoints

Please read GDB: Setting Watchpoints first.
Currently, VSCode does not provide an API to change the value format of watchpoints. You can manually cast watchpoint expressions to display the value as specific pointer types:

- \$pc, default decimal integer format
- 0x10012000, an address, default decimal integer format
- (void*)\$pc, \$pc register, hexadecimal format
- *(void**)0x10012000, an address, hexadecimal format

**Install Shell Commands**

Please refer to PlatformIO Core [Install Shell Commands](#).

**Proxy Server Support**

There are two options how to configure a proxy server:

1. Declare the `HTTP_PROXY` and `HTTPS_PROXY` system environment variables (for example `HTTP_PROXY=http://user:pass@10.10.1.10:3128/`, etc.)
2. Open VSCode Settings and search for “Proxy”. Please set “Http: Proxy” and disable “Http: Proxy Strict SSL”.

**Key Bindings**

- `ctrl+alt+b` / `cmd-shift-b` / `ctrl-shift-b` Build Project
- `cmd-shift-d` / `ctrl-shift-d` Debug project
- `ctrl+alt+u` Upload Firmware
- `ctrl+alt+s` Open Serial Port Monitor

You can override existing key bindings or add a new in VSCode. See official documentation [Key Bindings for Visual Studio Code](#).

**Settings**

How to configure VSCode settings?

**platformio-ide.activateOnlyOnPlatformIOProject**

If true, activate the platformio ide extension only when a PlatformIO-based project (that has a “platformio.ini” *(Project Configuration File)*) is open in the workspace. The default value is false.

**platformio-ide.autoCloseSerialMonitor**

If true, automatically close pio device monitor before uploading/testing. The default value is true.
platformio-ide.autoRebuildAutocompleteIndex

If true, automatically rebuild the C/C++ Project Index when “platformio.ini” (Project Configuration File) is changed or when new libraries are installed. The default value is true.

platformio-ide.buildTask

The build task (label) that is launched by the “Build” button in the PlatformIO Toolbar and Key Bindings. The default is PlatformIO: Build.

You can create custom Custom Tasks and assign one of them to platformio-ide.buildTask.

platformio-ide.autoPreloadEnvTasks

Automatically preload ALL project environment tasks. The default value is false.

platformio-ide.customPATH

Custom PATH for the platformio command. Paste here the result of echo $PATH (Unix) / echo %PATH% (Windows) command by typing into your system terminal if you prefer to use a custom version of PlatformIO Core (CLI). The default value is null, meaning PlatformIO looks for the platformio command in the system path.

platformio-ide.disableToolbar

Disable the PlatformIO toolbar. The default value is false.

platformio-ide.forceUploadAndMonitor

If true, the Upload (platformio-ide.upload) command is changed to use the “Upload and Monitor” task. The default value is false.

platformio-ide.reopenSerialMonitorDelay

Configure the time in milliseconds before reopening the Serial Port Monitor. The default value is 0, which means to reopen instantly.

platformio-ide.useBuiltinPython

Use a portable Python 3 Interpreter if available. The default value is true.

platformio-ide.useBuiltinPIOCore

If true, use the built-in PlatformIO Core (CLI). The default value is true.
platformio-ide.useDevelopmentPIOCore

If true, use the development version of PlatformIO Core (CLI). The default value is false.

platformio-ide.disablePIOHomeStartup

Disable showing PlatformIO Home at startup. The default value is false.

platformio-ide.pioHomeServerHttpPort

PlatformIO Home server HTTP port. The default value 0 automatically assigns a free port in the range [8010..8100]).

Known issues

PackageManager is unable to install tool

This is a known bug in VSCode Terminal issue #61.

A temporary solution is to install packages using a system terminal (not VSCode Terminal). Please use “Solution 3: Run from Terminal” in FAQ > Package Manager > [Error 5] Access is denied. Afterwards, go back to using the VSCode Terminal.

Changelog

Please visit the releases page.

1.20 Continuous Integration

Continuous Integration (CI, wiki) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day.

pio ci command is intended to be used in combination with the build servers and the popular Continuous Integration Software.

By integrating regularly, you can detect errors quickly, and locate them more easily.

1.20.1 AppVeyor

AppVeyor is an open-source hosted, distributed continuous integration service used to build and test projects hosted at GitHub on Windows family systems.

AppVeyor is configured by adding a file named appveyor.yml, which is a YAML format text file, to the root directory of the GitHub repository.

AppVeyor automatically detects when a commit has been made and pushed to a repository that is using AppVeyor, and each time this happens, it will try to build the project using pio ci command. This includes commits to all branches, not just to the master branch. AppVeyor will also build and run pull requests. When that process has completed, it will notify a developer in the way it has been configured to do so — for example, by sending an email containing the build results (showing success or failure), or by posting a message on an IRC channel. It can be configured to build project on a range of different Development Platforms.
Integration

Put appveyor.yml to the root directory of the GitHub repository. The contents of this file depends on the project you want to add. There are two possible ways of running PlatformIO in CI services:

Using pio run command

This variant is default choice for native PlatformIO projects:

```yaml
build: off
environment:

install:
  - cmd: git submodule update --init --recursive
  - cmd: SET PATH=%PATH%;C:\Python27\Scripts
  - cmd: pip install -U platformio

test_script:
  - cmd: pio run -e <ID_1> -e <ID_2> -e <ID_N>
```

Using pio ci command

This variant is more convenient when project is written as a library (when there are examples or testing code) as it has additional options for specifying extra libraries and boards from command line interface:

```yaml
build: off
environment:

matrix:
  - PLATFORMIO_CI_SRC: "path\to\source\file.c"
  - PLATFORMIO_CI_SRC: "path\to\source\file.ino"
  - PLATFORMIO_CI_SRC: "path\to\source\directory"

install:
  - cmd: git submodule update --init --recursive
  - cmd: SET PATH=%PATH%;C:\Python27\Scripts
  - cmd: pip install -U platformio

test_script:
  - cmd: pio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
```
Examples

1. Integration for USB_Host_Shield_2.0 project. The appveyor.yml configuration file:

```yaml
build: off
environment:

  matrix:
    - PLATFORMIO_CI_SRC: "examples\Bluetooth\PS3SPP\PS3SPP.ino"
    - PLATFORMIO_CI_SRC: "examples\pl2303\pl2303_gps\pl2303_gps.ino"

install:
  - cmd: git submodule update --init --recursive
  - cmd: SET PATH=%PATH%;C:\Python27\Scripts
  - cmd: pip install -U platformio
  - cmd: git clone https://github.com/xxxajk/spi4teensy3.git C:\spi4teensy

test_script:
  - cmd: pio ci --lib="." --lib="C:\spi4teensy" --board=uno --board=teensy31 --
     --board=due
```

1.20.2 CircleCI

CircleCI is a hosted cloud platform that provides hosted continuous integration, deployment, and testing to GitHub repositories.

CircleCI is configured by adding a file named circle.yml, which is a YAML format text file, to the root directory of the GitHub repository.

CircleCI automatically detects when a commit has been made and pushed to a repository that is using CircleCI, and each time this happens, it will try to build the project using pio ci command. This includes commits to all branches, not just to the master branch. CircleCI will also build and run pull requests. When that process has completed, it will notify a developer in the way it has been configured to do so — for example, by sending an email containing the build results (showing success or failure), or by posting a message on an IRC channel. It can be configured to build project on a range of different Development Platforms.

Contents

- CircleCI
  - Integration
    - Using pio run command
    - Using pio ci command
      - Library dependencies
    - Install dependent library using Library Management
    - Manually download dependent library and include in build process via --lib option
      - Custom Build Flags
      - Advanced configuration
  - Examples
Integration

Note: Please make sure to read CircleCI Getting Started guide first.

There are two possible ways of running PlatformIO in CI services:

**Using pio run command**

This variant is default choice for native PlatformIO projects:

```
dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio
  test:
    override:
    - pio run -e <ID_1> -e <ID_2> -e <ID_N>
```

**Using pio ci command**

This variant is more convenient when project is written as a library (when there are examples or testing code) as it has additional options for specifying extra libraries and boards from command line interface:

```
dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio
  test:
    override:
    - pio ci path/to/test/file.c --board=<ID_1> --board=<ID_2> --board=<ID_N>
    - pio ci examples/file.ino --board=<ID_1> --board=<ID_2> --board=<ID_N>
    - pio ci path/to/test/directory --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

**Library dependencies**

There 2 options to test source code with dependent libraries:

**Install dependent library using Library Management**

```
dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio
    # OneWire Library with ID=1 https://platformio.org/lib/show/1/OneWire
    - pio lib -g install 1
```

(continues on next page)
test:
  override:
    - pio ci path/to/test/file.c --board=<ID_1> --board=<ID_2> --board=<ID_N>

Manually download dependent library and include in build process via \texttt{--lib} option

dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio
    # download library to the temporary directory
    - wget https://github.com/PaulStoffregen/OneWire/archive/master.zip -O /tmp/onewire_source.zip
      - unzip /tmp/onewire_source.zip -d /tmp/
  test:
    override:
      - pio ci path/to/test/file.c --lib="/tmp/OneWire-master" --board=<ID_1> --board=<ID_2> --board=<ID_N>

Custom Build Flags

PlatformIO allows one to specify own build flags using \texttt{PLATFORMIO\_BUILD\_FLAGS} environment

machine:
  environment:
    \texttt{PLATFORMIO\_BUILD\_FLAGS}: \texttt{-D \texttt{SPECIFIC\_MACROS} -I/extra/inc}

For the more details, please follow to available build flags/options.

Advanced configuration

PlatformIO allows one to configure multiple build environments for the single source code using \texttt{“platformio.ini” (Project Configuration File)}.

Instead of \texttt{--board} option, please use \texttt{pio ci \texttt{--project-conf}}

Example:

1. Custom build flags

dependencies:
  cachedirectories:
    - "~/.platformio"
PlatformIO Documentation, Release 5.0.5a1

(continued from previous page)

pre:

- sudo pip install -U platformio
  
  # pre-install PlatformIO development platforms, they will be cached
  - pio platform install atmelavr atmelsam teensy

  #
  # Libraries from PlatformIO Library Registry:
  #
  # https://platformio.org/lib/show/416/TinyGPS
  # https://platformio.org/lib/show/417/SPI4Teensy3
  - pio lib -g install 416 417

  
  test:

  override:

  pio ci examples/acm/acm_terminal --board=uno --board=teensy31 --board=due --
  --lib="."
  - pio ci examples/adk/adk_barcode --board=uno --board=teensy31 --board=due --
  --lib="."
  - pio ci examples/adk/ArduinoBlinkLED --board=uno --board=teensy31 --
    --board=due --lib="."
  - pio ci examples/adk/demokit_20 --board=uno --board=teensy31 --board=due --
    --lib="."
    # ...
    # pio ci examples/Xbox/XBOXUSB --board=uno --board=teensy31 --board=due --
    --lib="."

2. Dependency on external libraries

dependencies:

pre:

  # Install the latest stable PlatformIO
  - sudo pip install -U platformio

  # download dependent libraries
  - wget https://github.com/jcw/jeelib/archive/master.zip -O /tmp/jeelib.zip
    - unzip /tmp/jeelib.zip -d /tmp
    - unzip /tmp/gamebuino.zip -d /tmp

  # ...
  
  test:

  override:

  - pio ci examples/backSoon/backSoon.ino --lib="."
    --lib="/tmp/jeelib-master"
    --lib="/tmp/Gamebuino-master/libraries/tinyFAT"
    --board=uno --board=megaatmega2560

  - pio ci examples/etherNode/etherNode.ino --lib="."
    --lib="/tmp/jeelib-master"
    --lib="/tmp/Gamebuino-master/libraries/tinyFAT"
    --board=uno --board=megaatmega2560

  # ...

(continues on next page)
1.20.3 Drone

Drone is a hosted continuous integration service. It enables you to conveniently set up projects to automatically build, test, and deploy as you make changes to your code to GitHub and BitBucket repositories.

Drone is configured by modifying settings in your project control panel.

Drone automatically detects when a commit has been made and pushed to a repository that is using Drone, and each time this happens, it will try to build the project using `pio ci` command. This includes commits to all branches, not just to the master branch. Drone will also build and run pull requests. When that process has completed, it will notify a developer in the way it has been configured to do so — for example, by sending an email containing the build results (showing success or failure). It can be configured to build project on a range of different Development Platforms.

### Contents

- **Drone**
  - Integration
    - Using `pio run` command
    - Using `pio ci` command
  - Examples

### Integration

There are two possible ways of running PlatformIO in CI services:

#### Using `pio run` command

This variant is default choice for native PlatformIO projects (Please fill all fields for your project in the Drone control panel):

**Commands:**

```
pip install -U platformio
pio run -e <ID_1> -e <ID_2> -e <ID_N>
```
Using pio ci command

This variant is more convenient when project is written as a library (when there are examples or testing code) as it has additional options for specifying extra libraries and boards from command line interface:

Environment Variables:

- `PLATFORMIO_CI_SRC=path/to/source/file.c`
- `PLATFORMIO_CI_SRC=path/to/source/file.ino`
- `PLATFORMIO_CI_SRC=path/to/source/directory`

Commands:

- `pip install -U platformio`
- `pio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>`
Examples

1. Integration for USB_Host_Shield_2.0 project. The `circle.yml` configuration file:

   Environment Variables:

   ```
   PLATFORMIO_CI_SRC=examples/Bluetooth/PS3SPP/PS3SPP.ino
   PLATFORMIO_CI_SRC=examples/pl2303/pl2303_gps/pl2303_gps.ino
   ```

   Commands:
1.20.4 GitHub Actions

GitHub Actions enables you to create custom software development life cycle (SDLC) workflows directly in your
GitHub repository.

You need to configure GitHub Actions using YAML syntax, and save them as workflow files in your repository. Workflows are custom automated processes that you can set up in your repository to build, test, package, release, or deploy any code project on GitHub. You can write individual tasks, called actions, and combine them to create a custom workflow. Once you’ve successfully created a YAML workflow file and triggered the workflow, you will see the build logs, tests results, artifacts, and statuses for each step of your workflow. It can be configured to build project on a range of different Development Platforms.

GitHub Actions help you automate your software development workflows in the same place you store code and collaborate on pull requests and issues and each time this happens, it will try to build the project using pio ci command.

## Contents

- **GitHub Actions**
  - Integration
    * Using pio run command
    * Using pio ci command
      - Library dependencies
    * Install dependent library using Library Management
    * Manually download dependent library and include in build process via --lib option
      - Custom Build Flags
    - Examples

## Integration

**Note:** Please make sure to read GitHub Actions Getting Started guide first.

There are two possible ways of running PlatformIO in CI services:

### Using pio run command

This variant is default choice for native PlatformIO projects:

```yaml
name: PlatformIO CI

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Cache pip
        uses: actions/cache@v2

```

(continues on next page)
with:
  path: ~/.cache/pip
  key: ${{ runner.os }}-pip-{{ hashFiles('**/requirements.txt') }}
  restore-keys: |
    ${{ runner.os }}-pip-
  - name: Cache PlatformIO
    uses: actions/cache@v2
    with:
      path: ~/.platformio
      key: ${{ runner.os }}-{{ hashFiles('**/lockfiles') }}
      restore-keys: ${{ runner.os }}-platformio-
  - name: Set up Python
    uses: actions/setup-python@v2
  - name: Install PlatformIO
    run: |
      python -m pip install --upgrade pip
      pip install --upgrade platformio
  - name: Run PlatformIO
    run: pio run -e <ID_1> -e <ID_2> -e <ID_N>

Using pio ci command

This variant is more convenient when project is written as a library (when there are examples or testing code) as it has additional options for specifying extra libraries and boards from command line interface:

```yaml
name: PlatformIO CI

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        example: [path/to/test/file.c, examples/file.ino, path/to/test/directory]

    steps:
    - uses: actions/checkout@v2
    - name: Cache pip
      uses: actions/cache@v2
      with:
        path: ~/.cache/pip
        key: ${{ runner.os }}-pip-{{ hashFiles('**/requirements.txt') }}
        restore-keys: ${{ runner.os }}-pip-
    - name: Cache PlatformIO
      uses: actions/cache@v2
      with:
        path: ~/.platformio
        key: ${{ runner.os }}-{{ hashFiles('**/lockfiles') }}
        restore-keys: ${{ runner.os }}-platformio-
    - name: Set up Python
      uses: actions/setup-python@v2
    - name: Install PlatformIO
      run: |
        python -m pip install --upgrade pip
        pip install --upgrade platformio
```

(continues on next page)
- name: Run PlatformIO
  run: pio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
  env:
    PLATFORMIO_CI_SRC: ${{ matrix.example }}

Library dependencies

There are two options to test source code with dependent libraries:

Install dependent library using Library Management

- name: Install library dependencies
  run: pio lib -g install 1

- name: Run PlatformIO
  run: pio ci path/to/test/file.c --board=<ID_1> --board=<ID_2> --board=<ID_N>

Manually download dependent library and include in build process via --lib option

- name: Install library dependencies
  run: |
    wget https://github.com/PaulStoffregen/OneWire/archive/master.zip -O /tmp/onewire_source.zip
    unzip /tmp/onewire_source.zip -d /tmp/

- name: Run PlatformIO
  run: pio ci path/to/test/file.c --lib="/tmp/OneWire-master" --board=<ID_1> --board=<ID_2> --board=<ID_N>

Custom Build Flags

PlatformIO allows one to specify own build flags using PLATFORMIO_BUILD_FLAGS environment

- name: Run PlatformIO
  run: pio ci path/to/test/file.c --lib="/tmp/OneWire-master" --board=<ID_1> --board=<ID_2> --board=<ID_N>
  env:
    PLATFORMIO_BUILD_FLAGS: -D SPECIFIC_MACROS -I/extra/inc

For the more details, please follow to available build flags/options.

Examples

Integration for USB_Host_Shield_2.0 project. The workflow.yml configuration file:

name: PlatformIO CI

on: [push]
jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, macos-latest, windows-latest]
        example: [examples/Bluetooth/PS3SPP/PS3SPP.ino, examples/pl2303/pl2303_gps/pl2303_gps.ino]
    steps:
      - uses: actions/checkout@v2
      - name: Cache pip
        uses: actions/cache@v2
        with:
          path: ~/.cache/pip
          key: ${{ runner.os }}-pip-${{ hashFiles('**/requirements.txt') }}
          restore-keys:
            - ${{ runner.os }}-pip-
      - name: Cache PlatformIO
        uses: actions/cache@v2
        with:
          path: ~/.platformio
          key: ${{ runner.os }}-${{ hashFiles('**/lockfiles') }}
      - name: Set up Python
        uses: actions/setup-python@v2
      - name: Install PlatformIO
        run: |
          python -m pip install --upgrade pip
          pip install --upgrade platformio
          unzip /tmp/spi4teensy3.zip -d /tmp
          pio ci --lib="." --lib="/tmp/spi4teensy3-master" --board=uno --board=teensy31 --board=due
          env:
            PLATFORMIO_CI_SRC: ${{ matrix.example }}

1.20.5 GitLab

GitLab is a hosted cloud platform that can help you build, test, deploy, and monitor your code from GitLab repositories. GitLab CI is enabled by default on new projects, so you can start using its features right away. All you need is `pio ci` command, a file called `.gitlab-ci.yml` (where you describe how the build should run) placed in the root directory of your git project, and a configured Runner to perform the actual build (Gitlab has some pre-configured public runners so your CI script should work out of the box). Each project comes with a Builds page where you can follow the output of each build, see the commit that introduced it and other useful information such as the time the build started, how long it lasted and the committer’s name. The statuses for each build are exposed in the GitLab UI, and you can see whether a build succeeded, failed, got canceled or skipped.
Integration

Put .gitlab-ci.yml to the root directory of your repository. The contents of this file depends on the project you want to add. There are two possible ways of running PlatformIO in CI services:

Using pio run command

This variant is default choice for native PlatformIO projects:

```yaml
image: python:2.7
stages:
  - test
before_script:
  - "pip install -U platformio"
job:
  stage: test
  script: "pio run -e <ID_1> -e <ID_2> -e <ID_N>"
```

Using pio ci command

This variant is more convenient when project is written as a library (when there are examples or testing code) as it has additional options for specifying extra libraries and boards from command line interface:

```yaml
image: python:2.7
stages:
  - test
before_script:
  - "pip install -U platformio"
job:
  stage: test
  script: "pio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>"
  variables: {
    PLATFORMIO_CI_SRC: "path/to/test/file.c"
  }
```
Examples

1. Integration for ArduinoJson library project. The .gitlab-ci.yml configuration file:

```
image: python:2.7

stages:
- test

.job_template: &pio_run
  script:
    - "pio ci --lib='.' --board=uno --board=teensy31 --board=nodemcu
      --$PLATFORMIO_CI_EXTRA_ARGS"

before_script:
  - "pip install -U platformio"

JsonGeneratorExample:
  <<: *pio_run
  variables:
    PLATFORMIO_CI_SRC: examples/JsonGeneratorExample

JsonHttpClient:
  <<: *pio_run
  variables:
    PLATFORMIO_CI_SRC: examples/JsonHttpClient

JsonParserExample:
  <<: *pio_run
  variables:
    PLATFORMIO_CI_SRC: examples/JsonParserExample

JsonServer:
  <<: *pio_run
  variables:
    PLATFORMIO_CI_SRC: examples/JsonServer

JsonUdpBeacon:
  <<: *pio_run
  variables:
    PLATFORMIO_CI_SRC: examples/JsonUdpBeacon

ProgmemExample:
  stage: test
  <<: *pio_run
  variables:
    PLATFORMIO_CI_SRC: examples/ProgmemExample

StringExample:
  stage: test
  <<: *pio_run
  variables:
    PLATFORMIO_CI_SRC: examples/StringExample
```
1.20.6 Jenkins

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

It can be configured to build project on a range of different Development Platforms.

Contents

- Jenkins
  - Integration

Integration

See step-by-step guide in ThingForward’s blog post Setting up a Jenkins CI engine for embedded projects.

1.20.7 Shippable

Shippable is a hosted cloud platform that provides hosted continuous integration, deployment, and testing to GitHub and BitBucket repositories. Shippable’s continuous integration service is built using Docker.

Shippable is configured by adding a file named shippable.yml, which is a YAML format text file, to the root directory of the GitHub repository or you can use your Travis CI configuration file .travis.yml.

Shippable automatically detects when a commit has been made and pushed to a repository that is using Shippable, and each time this happens, it will try to build the project using pio ci command. This includes commits to all branches, not just to the master branch. Shippable will also build and run pull requests. When that process has completed, it will notify a developer in the way it has been configured to do so — for example, by sending an email containing the build results (showing success or failure), or by posting a message on an IRC channel. It can be configured to build project on a range of different Development Platforms.

Contents

- Shippable
  - Integration
    - Using pio run command
    - Using pio ci command
  - Examples

Integration

Put shippable.yml or .travis.yml to the root directory of your repository. The contents of this file depends on the project you want to add. There are two possible ways of running PlatformIO in CI services:
Using pio run command

This variant is default choice for native PlatformIO projects:

```yaml
language: python
python:
  - "2.7"
install:
  - pip install -U platformio
script:
  - pio run -e <ID_1> -e <ID_2> -e <ID_N>
```

Using pio ci command

This variant is more convenient when project is written as a library (when there are examples or testing code) as it has additional options for specifying extra libraries and boards from command line interface:

```yaml
language: python
python:
  - "2.7"
env:
  - PLATFORMIO_CI_SRC=path/to/source/file.c
  - PLATFORMIO_CI_SRC=path/to/source/file.ino
  - PLATFORMIO_CI_SRC=path/to/source/directory
install:
  - pip install -U platformio
script:
  - pio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

Examples

1. Integration for USB_Host_Shield_2.0 project. The shippable.yml or .travis.yml configuration file:

```yaml
language: python
python:
  - "2.7"
env:
  - PLATFORMIO_CI_SRC=examples/Bluetooth/PS3SPP/PS3SPP.ino
  - PLATFORMIO_CI_SRC=examples/pl2303/pl2303_gps/pl2303_gps.ino
install:
  - pip install -U platformio
  - unzip /tmp/spi4teensy3.zip -d /tmp
script:
  - pio ci --lib="." --lib="/tmp/spi4teensy3-master" --board=uno --board=teensy31 --board=due
```
1.20.8 Travis CI

Travis CI officially supports PlatformIO for Embedded Builds.

Travis CI is an open-source hosted, distributed continuous integration service used to build and test projects hosted at GitHub.

Travis CI is configured by adding a file named `.travis.yml`, which is a YAML format text file, to the root directory of the GitHub repository.

Travis CI automatically detects when a commit has been made and pushed to a repository that is using Travis CI, and each time this happens, it will try to build the project using `pio ci` command. This includes commits to all branches, not just to the master branch. Travis CI will also build and run pull requests. When that process has completed, it will notify a developer in the way it has been configured to do so — for example, by sending an email containing the build results (showing success or failure), or by posting a message on an IRC channel. It can be configured to build project on a range of different Development Platforms.

Contents

- Travis CI
  - Integration
    * Using pio run command
    * Using pio ci command
  - Library dependencies
    * Install dependent library using Library Management
    * Manually download dependent library and include in build process via `--lib` option
  - Custom Build Flags
  - Advanced configuration
  - Unit Testing
  - Examples

Integration

Please make sure to read Travis CI Getting Started and general build configuration guides first.

Note: If you are going to use PlatformIO Unit Testing or Remote Development you will need to define `PLATFORMIO_AUTH_TOKEN` environment variable in project settings. See Defining Variables in Repository Set-
PlatformIO is written in Python and is recommended to be run within Travis CI Python isolated environment. There are two possible ways of running PlatformIO in CI services:

Using pio run command

This variant is default choice for native PlatformIO projects:

```yaml
language: python
python: - "3.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/.platformio"
    - $HOME/.cache/pip
install:
  - pip install -U platformio
  - pio update
script:
  - pio run -e <ID_1> -e <ID_2> -e <ID_N>
```

Using pio ci command

This variant is more convenient when project is written as a library (when there are examples or testing code) as it has additional options for specifying extra libraries and boards from command line interface:

```yaml
language: python
python: - "3.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/.platformio"
    - $HOME/.cache/pip

env:
  - PLATFORMIO_CI_SRC=path/to/test/file.c
  - PLATFORMIO_CI_SRC=examples/file.ino
  - PLATFORMIO_CI_SRC=path/to/test/directory

install:
  - pip install -U platformio
  - pio update

script:
  - pio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
```
Then perform steps 1, 2 and 4 from http://docs.travis-ci.com/user/getting-started/

Library dependencies

There are 2 options to test source code with dependent libraries:

Install dependent library using Library Management

```
install:
  - pip install -U platformio
  
  # Libraries from PlatformIO Library Registry:
  
  # https://platformio.org/lib/show/1/OneWire
  - pio lib -g install 1
```

Manually download dependent library and include in build process via --lib option

```
install:
  - pip install -U platformio
  
  # download library to the temporary directory
  - wget https://github.com/PaulStoffregen/OneWire/archive/master.zip -O /tmp/onewire_source.zip
  - unzip /tmp/onewire_source.zip -d /tmp/

script:
  - pio ci --lib="/tmp/OneWire-master" --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

Custom Build Flags

PlatformIO allows one to specify own build flags using `PLATFORMIO_BUILD_FLAGS` environment

```
env:
  - PLATFORMIO_CI_SRC=path/to/test/file.c PLATFORMIO_BUILD_FLAGS="-D SPECIFIC MACROS_PER_TEST_ENV -I/extra/inc"
  - PLATFORMIO_CI_SRC=examples/file.ino
  - PLATFORMIO_CI_SRC=path/to/test/directory

install:
  - pip install -U platformio
  
  export PLATFORMIO_BUILD_FLAGS="-D GLOBAL_MACROS_FOR_ALL_TEST_ENV"
```

For the more details, please follow to available build flags/options.

Advanced configuration

PlatformIO allows one to configure multiple build environments for the single source code using “platformio.ini” (Project Configuration File).

Instead of --board option, please use `pio ci --project-conf`
Unit Testing

See PlatformIO Remote Unit Testing Example.

Examples

1. Custom build flags

```yaml
language: python
python:
  - "3.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/.platformio"
    - $HOME/.cache/pip

env:
  - PLATFORMIO_CI_SRC=examples/acm/acm_terminal
  - PLATFORMIO_CI_SRC=examples/Bluetooth/WiiIRCamera PLATFORMIO_BUILD_FLAGS="-DWIIICAMERA"
  - PLATFORMIO_CI_SRC=examples/ftdi/USBFTDILoopback
  - PLATFORMIO_CI_SRC=examples/Xbox/XBOXUSB
    # - ...

install:
  - pip install -U platformio
  - pio update

# Libraries from PlatformIO Library Registry:
# https://platformio.org/lib/show/416/TinyGPS
# https://platformio.org/lib/show/417/SPI4Teensy3
  - pio lib -g install 416 417

script:
  - pio ci --board=uno --board=teensy3l --board=due --lib="."
```

- Configuration file: https://github.com/felis/USB_Host_Shield_2.0/blob/master/.travis.yml
- Build History: https://travis-ci.org/felis/USB_Host_Shield_2.0

2. Dependency on external libraries

```yaml
language: python
python:
  - "3.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
(continues on next page)
sudo: false

cache:
  directories:
    - "~/.platformio"
    - $HOME/.cache/pip

env:
  - PLATFORMIO_CI_SRC=examples/backSoon/backSoon.ino
  - PLATFORMIO_CI_SRC=examples/etherNode/etherNode.ino

install:
  - pip install -U platformio
  - pio update

  - wget https://github.com/jcw/jeelib/archive/master.zip -O /tmp/jeelib.zip
  - unzip /tmp/jeelib.zip -d /tmp

  - unzip /tmp/gamebuino.zip -d /tmp

script:
  - pio ci --lib="." --lib="/tmp/jeelib-master" --lib="/tmp/Gamebuino-master/
  - board=uno --board=megaatmega2560

  • Configuration file: https://github.com/jcw/ethercard/blob/master/.travis.yml
  • Build History: https://travis-ci.org/jcw/ethercard

  3. Dynamic testing of the boards

language: python

python:
  - "3.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure

# sudo: false

cache:
  directories:
    - "~/.platformio"
    - $HOME/.cache/pip

env:
  - PLATFORMIO_CI_SRC=examples/TimeArduinoDue PLATFORMIO_CI_EXTRA_ARGS="--board=due"
  - PLATFORMIO_CI_SRC=examples/TimeGPS
  - PLATFORMIO_CI_SRC=examples/TimeNTP
  - PLATFORMIO_CI_SRC=examples/TimeTeensy3 PLATFORMIO_CI_EXTRA_ARGS="--board=teensy31"
# ...

install:
  - pip install -U platformio
  - pio update
  - rm -rf ./linux
# (continues on next page)
# Libraries from PlatformIO Library Registry:
- https://platformio.org/lib/show/416/TinyGPS
- pio lib -g install 416 421 422

script:
- pio ci --lib="." --board=uno --board=teensy20pp $PLATFORMIO_CI_EXTRA_ARGS

- Configuration file: https://github.com/ivankravets/Time/blob/master/.travis.yml
- Build History: https://travis-ci.org/ivankravets/Time

4. Advanced configuration with extra project options and libraries

```yaml
language: python
python:
  - "3.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/.platformio"
    - "$HOME/.cache/pip

env:
- PLATFORMIO_CI_SRC=examples/Boards_Bluetooth/Adafruit_Bluefruit_LE PLATFORMIO_CI_EXTRA_ARGS="--board=genuino101"
- PLATFORMIO_CI_SRC=examples/Boards_Bluetooth/Arduino_101_BLE PLATFORMIO_CI_EXTRA_ARGS="--board=bluepill_f103c8 --project-option='framework=arduino'"
- PLATFORMIO_CI_SRC=examples/Boards_USB_Serial/Blue_Pill_STM32F103C PLATFORMIO_CI_EXTRA_ARGS="--board=bluepill_f103c8 --project-option='framework=arduino'"
- PLATFORMIO_CI_SRC=examples/Export_Demo/myPlant_ESP8266 PLATFORMIO_CI_EXTRA_ARGS="--board=nodemcu2 --project-option='lib_ignore=WiFi101'"
- ...

install:
- pip install -U platformio
- pio update

# Libraries from PlatformIO Library Registry:

- https://platformio.org/lib/show/44/Time
- https://platformio.org/lib/show/419/SimpleTimer
- https://platformio.org/lib/show/17/Adafruit-CC3000
- https://platformio.org/lib/show/28/SPI4Teensy3
- https://platformio.org/lib/show/91/UIPEthernet
- https://platformio.org/lib/show/418/WildFireCore
- https://platformio.org/lib/show/420/WildFire-CC3000
- https://platformio.org/lib/show/65/WiFlyHQ
- https://platformio.org/lib/show/19/Adafruit-DHT
- https://platformio.org/lib/show/299/WiFi101
- https://platformio.org/lib/show/259/BLEPeripheral
- https://platformio.org/lib/show/177/Adafruit_BluefruitLE_nRF51
```

(continues on next page)


**script:**
- make travis-build

- Configuration file: https://github.com/blynkkk/blynk-library/blob/master/.travis.yml
- Build History: https://travis-ci.org/blynkkk/blynk-library

### 1.21 Compilation database compile_commands.json

A compilation database is a JSON-formatted file named `compile_commands.json` that contains structured data about every compilation unit in your project. **PlatformIO Core (CLI)** supports generating of compilation database using `pio run --target` command and `compiledb` target. For example,

```
> pio run -t compiledb
```

A default path for `compile_commands.json` is “`build_dir/envname`”. You can override this path with `Advanced Scripting` and `COMPILATIONDB_PATH` environment variable. For example, generate `compile_commands.json` in a root of project:

```
platformio.ini:

[env:myenv]
platform = ...
board = ...
extra_scripts = post:extra_script.py
```

```
extra_script.py:

import os
Import("env")

env.Replace(COMPILATIONDB_PATH=os.path.join("$PROJECT_DIR", "compile_commands.json"))
```

### 1.22 Articles about us

**Note:** If you’ve written article about PlatformIO and would like it listed on this page, please edit this page.

Here are recent articles/reviews about PlatformIO:

#### 1.22.1 2020

- Aug 7, 2020 - Benjamin Cabé - Connecting the Wio Terminal to Azure IoT
- Jul 15, 2020 - Andreas Schmidt - IoT für Webentwickler: Unit Testing mit PlatformIO
- Jun 22, 2020 - Noud van Kruysbergen - RISC-V programmeren: met platform IO en microcontrollerboard
1.22.2 2019

- Dec 29, 2019 - Andri Yadi - AI-Powered Magic Wand
- Dec 2, 2019 - Wezley Sherman - TensorFlow, Meet The ESP32
- Oct 31, 2019 - Frank Leon Rose - Minimal FreeRTOS with PlatformIO
- Aug 18, 2019 - Manuel Bleichenbacher - Arduino In-circuit Debugging with PlatformIO
- Aug 13, 2019 - Tech Explorations - 6 reasons why PlatformIO is perhaps the best programming environment for the ESP32
- Jul 04, 2019 - Jean-Claude Wippler - The PlatformIO command line
- Mar 08, 2019 - Nathan Glover - Amazon Alexa controlled IoT Traffic Lights

1.22.3 2018

- Dec 27, 2018 - Xose Pérez - Automated unit testing in the metal
- Dec 20, 2018 - Jean-Claude Wippler - Getting started with the STM32F407VG and STM32Cube
- Dec 14, 2018 - Alvaro Viebrantz - Serverless Continuous Integration and OTA update flow for IoT devices using Google Cloud Build and Arduino
- Nov 24, 2018 - Martin Fasani - PlatformIO: An alternative to Arduino IDE and a complete ecosystem for IoT
- Sep 27, 2018 - Lup Yuen Lee - Connect STM32 Blue Pill to Sigfox
- Aug 27, 2018 - Lup Yuen Lee - Juggling STM32 Blue Pill For Arduino Jugglers
- Jul 3, 2018 - Andreas Schmidt - IoT for web developers: From zero to firmware, Part II
- Jun 22, 2018 - Andreas Schmidt - IoT for web developers, Part I
- Jul 4, 2018 - ThingForward - Screen-cast: First steps with PlatformIO’s Unified Debugger
- Jun 6, 2018 - Andreas Schmidt - ESP32, WebThing API and PlatformIO-style projects
• May 21, 2018 - Dentella Luca - ESP32, PlatformIO
• Mar 27, 2018 - Andreas Schmidt - Building a Web Of Things REST-API on an Arduino MKR1000 with PlatformIO
• Mar 19, 2018 - ThingForward - Webinar: Unit Testing for Embedded with PlatformIO and Qt Creator
• Feb 16, 2018 - Alex Corvis - DIY Virtual alike NEST Thermostat with Node-RED
• Jan 24, 2018 - ThingForward - Embedded and Cloud - Continuous Integration
• Jan 14, 2018 - IT4nextgen - 5 Best Development Software(IDE) for Internet of Things (IOT) in 2018
• Jan 13, 2018 - Rui Marinho - Quick start guide to flashing ESP8266-based devices with PlatformIO

1.22.4 2017

• Dec 26, 2017 - Coyt Barringer - Programming STM32F103 Blue Pill using USB Bootloader and PlatformIO
• Dec 1, 2017 - ThingForward - Using Cloud IDEs for Embedded Development: CodeAnywhere
• Nov 13, 2017 - ThingForward - Automating Static and Dynamic Code Analysis with PlatformIO
• Oct 18, 2017 - ThingForward - Getting Started with SigFox and PlatformIO
• Sep 18, 2017 - ThingForward - Embedded Testing with PlatformIO – Part 4: Continuous Integration
• Sep 06, 2017 - ThingForward - Embedded Testing with PlatformIO – Part 3: Remoting
• Sep 04, 2017 - Dror Gluska - Looking To The IoT Future With PlatformIO And ESP32
• Aug 23, 2017 - - Develop ESP32 With PlatformIO IDE
• Aug 08, 2017 - ThingForward - Embedded Testing with PlatformIO - Part 2
• Jul 25, 2017 - ThingForward - Start Embedded Testing with PlatformIO
• Jun 23, 2017 - Naresh Krish - Home Automation Using Wiscore, OpenHab and PlatformIO
• Jun 05, 2017 - Projects DIY - Démarrer avec PlatformIO IDE alternatif pour Arduino, ESP8266, ESP32 et autres micro-contrôleurs (Start with PlatformIO alternative IDE for Arduino, ESP8266, ESP32 and other micro-controllers, French)
• Apr 12, 2017 - Jane Elizabeth - Let’s talk IoT: PlatformIO puts developers back in the driver’s seat
• Apr 07, 2017 - Al Williams - Hackaday: PlatformIO and Visual Studio take over the world
• Mar 13, 2017 - Ryan Mulligan - Continuous testing for Arduino libraries using PlatformIO and Travis CI
• Feb 23, 2017 - Bastiaan Visee - Using PlatformIO for your Arduino projects
• Jan 12, 2017 - Tiest van Gool - OTA: PlatformIO and ESP8266

1.22.5 2016

• Dec 13, 2016 - Dr. Patrick Mineault - Multi-Arduino projects with PlatformIO
• Dec 08, 2016 - Cuong Tran Viet - PlatformIO is a solution
• Nov 10, 2016 - PiGreek - PlatformIO the new Arduino IDE ?!
• Oct 31, 2016 - Ricardo Quesada - Retro Challenge: announcing Commodore Home
• Oct 3, 2016 - Xose Pérez - Using the new Bean Loader CLI from PlatformIO
• Sep 20, 2016 - The Linux Foundation - 21 Open Source Projects for IoT
Chapter 1. Contents
• May 01, 2016 - Pedro Minatel - PlatformIO – Uma alternativa ao Arduino IDE (PlatformIO - An alternative to the Arduino IDE, Portuguese)

• Apr 23, 2016 - Al Williams - Hackaday: Atomic Arduino (and Other) Development

• Apr 16, 2016 - Satthittham Sangthong - PlatformIO Arduino IDE (Let’s play together with PlatformIO IDE [alternative to Arduino IDE], Thai)

• Apr 15, 2016 - Daniel Eichhorn - ESP8266: Offline Debugging with the Platformio Environment

• Apr 11, 2016 - Matjaz Trcek - Top 5 Arduino integrated development environments

• Apr 06, 2016 - Aleks - PlatformIO ausprobiert ( Tried PlatformIO, German)

• Apr 02, 2016 - Diego Pinto - Você tem coragem de abandonar a IDE do Arduino? PlatformIO + Atom (Do you dare to leave the Arduino IDE? PlatformIO + Atom, Portuguese)

• Mar 30, 2016 - Brandon Cannaday - Getting Started with PlatformIO and ESP8266 NodeMcu

• Mar 29, 2016 - Pablo Peñalve - PlatformIO + Geany + Raspberry Pi, Spanish

• Mar 24, 2016 - NAzT - PlatformIO Arduino Library (PlatformIO and advanced development for Arduino Library, Thai)

• Mar 16, 2016 - Jakub Skořepa - Instalace PlatformIO (PlatformIO IDE Installation, Czech)

• Mar 12, 2016 - Peter Marks - PlatformIO, the Arduino IDE for programmers

• Mar 12, 2016 - Richard Arthurs - Getting Started With PlatformIO

• Mar 07, 2016 - Joran Jessurun - New world with PlatformIO (New world with PlatformIO, Dutch)

• Mar 05, 2016 - brichacek.net - PlatformIO – otevřený ekosystém pro vývoj IoT (PlatformIO – an open source ecosystem for IoT development, Czech)

• Mar 04, 2016 - Ricardo Vega - Programa tu Arduino desde Atom (Program your Arduino from Atom, Spanish)

• Feb 28, 2016 - Alex Bloggt - PlatformIO vorgestellt (Introduction to PlatformIO IDE, German)

• Feb 25, 2016 - NutDIY - PlatformIO Blink On Nodemcu Dev Kit V1.0 (Thai)

• Feb 23, 2016 - Ptarmigan Labs - ESP8266 Over The Air updating – what are the options?

• Feb 22, 2016 - Grzegorz Holdys - How to Integrate PlatformIO with Netbeans

• Feb 19, 2016 - Embedds - Develop easier with PlatformIO ecosystem

• Feb 13, 2016 - Robert Cudmore - Programming an arduino with PlatformIO

• Jan 24, 2016 - Sergey Prilukin - How to use IntelliJ IDEA to develop and upload software for micro controllers like Arduino

• Jan 16, 2016 - Dani Eichhorn - ESP8266 Arduino IDE Alternative: PlatformIO

• Jan 11, 2016 - David Mills, Ph.D. - STM NUCLEOF401RE TIMER IO

• Jan 05, 2016 - Julien Rodrigues - Internet Of Things: The IDE scandal

1.22.6 2015

• Dec 22, 2015 - Jan Penninkhof - Over-the-Air ESP8266 programming using PlatformIO

• Dec 15, 2015 - stastaka - PlatformIO (Use a custom board for PlatformIO, Japanese)

• Dec 08, 2015 - Piotr Król - Using PlatformIO with TI MSP430 LunchPads

• Dec 01, 2015 - Michal Seroczyński - Push Notification from Arduino Yún with motion sensor
• Dec 01, 2015 - JetBrains CLion Blog - C++ Annotated: Fall 2015. Arduino Support in CLion using PlatformIO
• Dec 01, 2015 - Tateno Yuichi - ESP8266 CUI (Develop a ESP8266 in CUI, Japanese)
• Nov 29, 2015 - Keith Hughes - Using PlatformIO for Embedded Projects
• Nov 22, 2015 - Michał Seroczyński - Using PlatformIO to get started with Arduino in CLion IDE
• Nov 09, 2015 - ÁLvaro García Gómez - Programar con Arduino “The good way” (Programming with Arduino “The good way”, Spanish)
• Nov 06, 2015 - nocd5 - PlatformIOembedSTM32 Nucleomruby (Use mruby in the offline build for STM32 Nucleo board with mbed and PlatformIO, Japanese)
• Oct 21, 2015 - Vittorio Zaccaria - Using a cheap STM32 Nucleo to teach remote sensor monitoring
• Oct 18, 2015 - Nico Coetzee - First Arduino I2C Experience with PlatformIO
• Oct 10, 2015 - Floyd Hilton - Programming Arduino with Atom
• Oct 01, 2015 - Mistan - Compile and Upload Arduino Sketch with PlatformIO for Raspberry Pi Running Arch Linux
• Sep 30, 2015 - Jay Wiggins - PlatformIO Investigation
• Sep 01, 2015 - Thomas P. Weldon, Ph.D. - Improvised MBED FRDM-K64F Eclipse/PlatformIO Setup and Software Installation
• Aug 08, 2015 - Josh Glendenning - Armstrap Eagle and PlatformIO
• Aug 01, 2015 - Russell Davis - PlatformIO on the Raspberry Pi
• Jul 25, 2015 - DinoTools - Erste Schritte mit PlatformIO (Getting Started with PlatformIO, German)
• Jul 20, 2015 - Eli Fatsi - Arduino Development in Atom Editor
• Jul 14, 2015 - ElbinarIO - Programar para Arduino y otros microcontroladores desde la linea de comandos (Program Arduino and other microcontrollers from the command line, Spanish)
• Jul 11, 2015 - TrojanC - Learning Arduino GitHub Repository
• Jul 07, 2015 - Sho Hashimoto - PlatformIOArduino(Arduino development in PlatformIO, Japanese)
• Jun 02, 2015 - Alejandro Guirao Rodríguez - Discovering PlatformIO: The RaspberryPi / Arduino combo kit is a winner option when prototyping an IoT-style project
• May 17, 2015 - S.S - Arduino : vim + platformio (Arduino development at the command line: VIM + PlatformIO, Japanese)
• May 11, 2015 - IT Hare - From Web Developer to Embedded One: Interview with Ivan Kravets, The Guy Behind PlatformIO. Part II
• May 4, 2015 - IT Hare - From Web Developer to Embedded One: Interview with Ivan Kravets, The Guy Behind PlatformIO. Part I
• Apr 17, 2015 - Michael Ball - PlatformIO - A Cross-Platform Code Builder and Missing Library Manager
• Mar 23, 2015 - Atmel - Cross-board and cross-vendor embedded development with PlatformIO
• Mar 22, 2015 - Mark VandeWettering - Discovered a new tool for embedded development: PlatformIO
• Feb 25, 2015 - Hendrik Putzek - Use your favourite IDE together with Arduino
1.22.7 2014

• Oct 7, 2014 - Ivan Kravets, Ph.D. - Integration of PlatformIO library manager to Arduino and Energia IDEs
• Jun 20, 2014 - Ivan Kravets, Ph.D. - Building and debugging Atmel AVR (Arduino-based) project using Eclipse IDE+PlatformIO
• Jun 17, 2014 - Ivan Kravets, Ph.D. - How was PlatformIO born or why I love Python World

1.23 Frequently Asked Questions

Note: We have a big database with Frequently Asked Questions in our Community Forums. Please have a look at it.

Contents

• General
  – What is PlatformIO?
  – What is .pio directory
  – What is .pioenvs directory
  – Command completion in Terminal
• Install Python Interpreter
• Convert Arduino file to C++ manually
• Program Memory Usage
• Advanced Serial Monitor with UI
• Troubleshooting
  – Installation
    • Multiple PlatformIO Cores in a system
    • 'platformio' is not recognized as an internal or external command
    • 99-platformio-udev.rules
    • ImportError: cannot import name _remove_dead_weakref
  – Package Manager
    • [Error 5] Access is denied
      • Solution 1: Remove folder
      • Solution 2: Antivirus
      • Solution 3: Run from Terminal
      • Solution 4: Manual
  – Building
    • UnicodeWarning: Unicode equal comparison failed
    • UnicodeDecodeError: Non-ASCII characters found in build environment
1.23.1 General

What is PlatformIO?

Please refer to What is PlatformIO?

What is .pio directory

Please refer to workspace_dir.

What is .pioenvs directory

Please refer to build_dir.

Command completion in Terminal

Please refer to PlatformIO Shell Completion.

1.23.2 Install Python Interpreter

PlatformIO Core (CLI) is written in Python that is installed by default on the all popular OS except Windows. Please navigate to official website and Download the latest Python and install it. Please READ NOTES BELOW.

macOS Please read the “Important Information” displayed during installation for information about SSL/TLS certificate validation and the running the “Install Certificates.command”.

If you do not install SSL/TLS certificates, PlatformIO will not be able to download dependent packages, libraries, and toolchains.

Windows Please select Add Python to Path (see below), otherwise, python command will not be available.
1.23.3 Convert Arduino file to C++ manually

Some Cloud & Desktop IDE doesn’t support Arduino files (*.ino and .pde) because they are not valid C/C++ based source files:

1. Missing includes such as `#include <Arduino.h>`
2. Function declarations are omitted.

In this case, code completion and code linting do not work properly or are disabled. To avoid this issue you can manually convert your INO files to CPP.

For example, we have the next Demo.ino file:

```cpp
void setup () {
    someFunction(13);
}

void loop () {
    delay(1000);
}

void someFunction(int num) {
}
```

Let’s convert it to Demo.cpp:

1. Add `#include <Arduino.h>` at the top of the source file
2. Declare each custom function (excluding built-in, such as `setup` and `loop`) before it will be called.

The final Demo.cpp:

```cpp
#include <Arduino.h>

void someFunction(int num);
```

(continues on next page)
void setup () {
    someFunction(13);
}

void loop () {
    delay(1000);
}

void someFunction(int num) {
}

Finish.

1.23.4 Program Memory Usage

PlatformIO calculates firmware/program memory usage based on the next segments:

- **.text**: The code segment, also known as a text segment or simply as text, is where a portion of an object file or the corresponding section of the program’s virtual address space that contains executable instructions is stored and is generally read-only and fixed size.

- **.data**: The .data segment contains any global or static variables which have a pre-defined value and can be modified. The values for these variables are initially stored within the read-only memory (typically within .text) and are copied into the .data segment during the start-up routine of the program. Example,

  ```
  int val = 3;
  char string[] = "Hello World";
  ```

- **.bss**: Uninitialized data is usually adjacent to the data segment. The BSS segment contains all global variables and static variables that are initialized to zero or do not have explicit initialization in the source code. For instance, a variable defined as `static int i;` would be contained in the BSS segment.

The rough calculation could be done as:

- PROGRAM (Flash) = .text + .data
- DATA (RAM) = .bss + .data

If you need to print all memory sections and addresses, please use `pio run --verbose` command.

Recommended for reading:

- text, data and bss: Code and Data Size Explained

1.23.5 Advanced Serial Monitor with UI

PlatformIO Core provides CLI version (`pio device monitor`) of Serial Monitor. If you need advanced instrument with a rich UI, we recommend free and multi-platform CoolTerm serial port terminal application.
**Warning:** Please note that you need to manually disconnect (close serial port connection) in CoolTerm before doing uploading in PlatformIO. PlatformIO can not disconnect/connect to a target device automatically when CoolTerm is used.

### 1.23.6 Troubleshooting

#### Installation

**Multiple PlatformIO Cores in a system**

Multiple standalone PlatformIO Core (CLI) in a system could lead to the different issues. We highly recommend to keep one instance of PlatformIO Core or use built-in PlatformIO Core in PlatformIO IDE:

- **PlatformIO IDE for Atom** - Menu PlatformIO: Settings > PlatformIO IDE > Use built-in PlatformIO Core
- **VSCode - Settings** > Set `platformio-ide.useBuiltinPIOCore` to `true`.

Finally, if you have a standalone PlatformIO Core (CLI) in a system, please open system Terminal (not PlatformIO IDE Terminal) and uninstall obsolete PlatformIO Core:

```
pip uninstall platformio
```

If you used macOS "brew"

```
brew uninstall platformio
```

If you need to have PlatformIO Core (CLI) globally in a system, please Install Shell Commands.

#### 'platformio' is not recognized as an internal or external command

If you use PlatformIO IDE, please check in PlatformIO IDE Settings that “Use built-in PlatformIO Core” is enabled.

If you modify system environment variable `PATH` in your Bash/Fish/ZSH profile, please do not override global `PATH`. This line `export PATH="/my/custom/path"` is incorrect. Use `export PATH="/my/custom/path":$PATH` instead.

#### 99-platformio-udev.rules


**Note:** Please check that your board’s PID and VID are listed in the rules. You can list connected devices and their PID/VID using `pio device list` command.

This file must be placed at `/etc/udev/rules.d/99-platformio-udev.rules` (preferred location) or `/lib/udev/rules.d/99-platformio-udev.rules` (required on some broken systems).

Please open system Terminal and type
# Recommended

```bash
```

# OR, manually download and copy this file to destination folder

```bash
```

Restart “udev” management tool:

```bash
sudo service udev restart
# or
sudo udevadm control --reload-rules
sudo udevadm trigger
```

Ubuntu/Debian users may need to add own “username” to the “dialout” group if they are not “root”, doing this issuing:

```bash
sudo usermod -a -G dialout $USER
sudo usermod -a -G plugdev $USER
```

Similarly, Arch users may need to add their user to the “uucp” group

```bash
sudo usermod -a -G uucp $USER
sudo usermod -a -G lock $USER
```

**Note:** You will need to log out and log back in again (or reboot) for the user group changes to take effect.

After this file is installed, physically unplug and reconnect your board.

**ImportError: cannot import name _remove_dead_weakref**

Windows users can experience this issue when multiple Python interpreters are installed in a system and conflict each other. The easy way to fix this problem is uninstalling all Python interpreters using Windows Programs Manager and installing them manually again.

2. Install the latest Python interpreter, see [Install Python Interpreter](https://www.platformio.org/docs/integration/python.html) guide
3. Remove `C:\Users\YourUserName\platformio` and `C:\.platformio` folders if exist (do not forget to replace “YourUserName” with the real user name)
4. Restart *PlatformIO IDE*.

**Package Manager**

**[Error 5] Access is denied**

PlatformIO installs all packages to “`core_dir/packages`” directory. **You MUST HAVE** write access to this folder. Please note that *PlatformIO does not require* “*sudo*/administrative privileges.
Solution 1: Remove folder

A quick solution is to remove “core_dir/packages” folder and repeat installation/building/uploading again.

Solution 2: Antivirus

Some antivirus tools forbid programs to create files in the background. PlatformIO Package Manager does all work in the background: downloads package, unpacks archive in temporary folder and moves final files to “core_dir/packages” folder.

Antivirus tool can block PlatformIO, that is why you see “[Error 5] Access is denied”. Try to disable it for a while or add core_dir directory to exclusion/whitelist.

Solution 3: Run from Terminal

As we mentioned in “Solution 2”, antivirus tools can block background file system operations. Another solution is to run PlatformIO Core (CLI) from a system terminal.

1. Open System Terminal, on Windows cmd.exe (not PlatformIO IDE Terminal)
2. Build a project and upload firmware using PlatformIO Core (CLI) which will download and install all dependent packages:

```
# Change directory to PlatformIO Project where is located "platformio.ini"
$ cd path/to/platformio/project

# Force PlatformIO to install PlatformIO Home dependencies
$ pio home

# Force PlatformIO to install toolchains
$ pio run --target upload
```

If “pio” command is not globally available in your environment and you use PlatformIO IDE, please use built-in PlatformIO Core (CLI) which is located in:

- Windows: `C:\Users\{username}\.platformio\penv\Scripts\platformio` Please replace `{username}` with a real user name
- Unix: `~/.platformio/penv/bin/platformio`

Note: You can add platformio and pio commands to your system environment. See Install Shell Commands.
Solution 4: Manual

If none of the solutions above do work for you, you can download and unpack all packages manually to “core_dir/packages”.

Please visit PlatformIO Package Storage and download a package for your platform. A correct package path is “core_dir/packages/{package_name}/package.json”.

Building

UnicodeWarning: Unicode equal comparison failed

Full warning message is “UnicodeWarning: Unicode equal comparison failed to convert both arguments to Unicode - interpreting them as being unequal”.

KNOWN ISSUE. Please move your project to a folder which full path does not contain non-ASCII chars.

UnicodeDecodeError: Non-ASCII characters found in build environment

KNOWN ISSUE. PlatformIO Core (CLI) currently does not support projects which contain non-ASCII characters (codes) in a full path or depend on the libraries which use non-ASCII characters in their names.

TEMPORARY SOLUTION

1. Use PlatformIO IDE, it will automatically install PlatformIO Core (CLI) in a root of system disk (%DISK%/platformio) and avoid an issue when system User contains non-ASCII characters
2. Do not use non-ASCII characters in project folder name or its parent folders.

Also, if you want to place PlatformIO Core (CLI) in own location, see:

• Set PLATFORMIO_CORE_DIR environment variable with own path
• Configure custom location per project using core_dir option in “platformio.ini” (Project Configuration File).

ARM toolchain: cc1plus: error while loading shared libraries

See related answers for error while loading shared libraries.

Archlinux: libncurses.so.5: cannot open shared object file

Answered in issue #291.

Monitoring a serial port breaks upload

Answered in issue #384.
1.24 Release Notes

1.24.1 PlatformIO Core 5

A professional collaborative platform for embedded development

5.0.4 (2020-12-??)

- Check for debug server’s “ready_pattern” in “stderr”

5.0.4 (2020-12-30)

- Added “Core” suffix when showing PlatformIO Core version using `pio --version` command
- Improved `.ccls` configuration file for Emacs, Vim, and Sublime Text integrations
- Updated analysis tools:
  - Cppcheck v2.3 with improved C++ parser and several new MISRA rules
  - PVS-Studio v7.11 with new diagnostics and updated mass suppression mechanism
- Show a warning message about deprecated support for Python 2 and Python 3.5
- Do not provide “intelliSenseMode” option when generating configuration for VSCode C/C++ extension
- Fixed a “git-sh-setup: file not found” error when installing project dependencies from Git VCS (issue #3740)
- Fixed an issue with package publishing on Windows when Unix permissions are not preserved (issue #3776)

5.0.3 (2020-11-12)

- Added an error selector for Sublime Text build runner (issue #3733)
- Generate a working “projectEnvName” for PlatformIO IDE’s debugger for VSCode
- Force VSCode’s intelliSenseMode to “gcc-x64” when GCC toolchain is used
- Print ignored test suites and environments in the test summary report only in verbose mode (issue #3726)
- Fixed an issue when the package manager tries to install a built-in library from the registry (issue #3662)
- Fixed an issue when `pio package pack` ignores some folders (issue #3730)

5.0.2 (2020-10-30)

- Initialize a new project or update the existing passing working environment name and its options (issue #3686)
- Automatically build PlatformIO Core extra Python dependencies on a host machine if they are missed in the registry (issue #3700)
- Improved “core.call” RPC for PlatformIO Home (issue #3671)
- Fixed a “PermissionError: [WinError 5]” on Windows when an external repository is used with `lib_deps` option (issue #3664)
- Fixed a “KeyError: ‘versions’” when dependency does not exist in the registry (issue #3666)
• Fixed an issue with GCC linker when “native” dev-platform is used in pair with library dependencies (issue #3669)
• Fixed an “AssertionError: ensure_dir_exists” when checking library updates from simultaneous subprocesses (issue #3677)
• Fixed an issue when pio package publish command removes original archive after submitting to the registry (issue #3716)
• Fixed an issue when multiple pio lib install command with the same local library results in duplicates in lib_deps (issue #3715)
• Fixed an issue with a “wrong” timestamp in device monitor output using “time” filter (issue #3712)

5.0.1 (2020-09-10)

• Added support for “owner” requirement when declaring dependencies using library.json
• Fixed an issue when using a custom git/ssh package with platform_packages option (issue #3624)
• Fixed an issue with “ImportError: cannot import name ‘_get_backend’ from ‘cryptography.hazmat.backends’” when using Remote Development on RaspberryPi device (issue #3652)
• Fixed an issue when pio package unpublish command crashes (issue #3660)
• Fixed an issue when the package manager tries to install a built-in library from the registry (issue #3662)
• Fixed an issue with incorrect value for C++ language standard in IDE projects when an in-progress language standard is used (issue #3653)
• Fixed an issue with “Invalid simple block (semantic_version)” from library dependency that refs to an external source (repository, ZIP/Tar archives) (issue #3658)
• Fixed an issue when can not remove update or remove external dev-platform using PlatformIO Home (issue #3663)

5.0.0 (2020-09-03)

Please check Migration guide from 4.x to 5.0.

• Integration with the new PlatformIO Trusted Registry
  – Enterprise-grade package storage with high availability (multi replicas)
  – Secure, fast, and reliable global content delivery network (CDN)
  – Universal support for all packages:
    • Libraries
    • Development platforms
    • Toolchains
  – Built-in fine-grained access control (role-based, teams, organizations)
  – New CLI commands:
    • pio package – manage packages in the registry
    • pio access – manage package access for users, teams, and maintainers
• Integration with the new Account Management System
  – Manage organizations
Manage teams and team memberships

New Package Management System

- Integrated PlatformIO Core with the new PlatformIO Registry
- Support for owner-based dependency declaration (resolves name conflicts) (issue #1824)
- Automatically save dependencies to "platformio.ini" when installing using PlatformIO CLI (issue #2964)
- Follow SemVer complaint version constraints when checking library updates (issue #1281)
- Dropped support for "packageRepositories" section in "platform.json" manifest (please publish packages directly to the registry)

Build System

- Upgraded build engine to the SCons 4.0 - a next-generation software construction tool
  - Configuration files are Python scripts – use the power of a real programming language to solve build problems
  - Built-in reliable and automatic dependency analysis
  - Improved support for parallel builds
  - Ability to share built files in a cache to speed up multiple builds
- New Custom Targets
  - Pre/Post processing based on dependent sources (another target, source file, etc.)
  - Command launcher with own arguments
  - Launch command with custom options declared in "platformio.ini"
  - Python callback as a target (use the power of Python interpreter and PlatformIO Build API)
  - List available project targets (including dev-platform specific and custom targets) with a new pio run --list-targets command (issue #3544)
- Enable “cyclic reference” for GCC linker only for the embedded dev-platforms (issue #3570)
- Automatically enable LDF dependency chain+ mode (evaluates C/C++ Preprocessor conditional syntax) for Arduino library when “library.property” has “depends” field (issue #3607)
- Fixed an issue with improper processing of source files added via multiple Build Middlewares (issue #3531)
- Fixed an issue with the clean target on Windows when project and build directories are located on different logical drives (issue #3542)

Project Management

- Added support for "globstar/**" (recursive) pattern for the different commands and configuration options (pio ci, src_filter, check_patterns, library.json > srcFilter). Python 3.5+ is required
- Added a new -e, --environment option to pio project init command that helps to update a PlatformIO project using the existing environment
- Dump build system data intended for IDE extensions/plugins using a new pio project data command
- Do not generate ".travis.yml” for a new project, let the user have a choice

Unit Testing

- Updated PIO Unit Testing support for Mbed framework and added compatibility with Mbed OS 6
- Fixed an issue when running multiple test environments (issue #3523)
– Fixed an issue when Unit Testing engine fails with a custom project configuration file (issue #3583)

• **Static Code Analysis**
  – Updated analysis tools:
    * Cppcheck v2.1 with a new “soundy” analysis option and improved code parser
    * PVS-Studio v7.09 with a new file list analysis mode and an extended list of analysis diagnostics
  – Added Cppcheck package for ARM-based single-board computers (issue #3559)
  – Fixed an issue with PIO Check when a defect with a multiline error message is not reported in verbose mode (issue #3631)

• **Miscellaneous**
  – Display system-wide information using a new `pio system info` command (issue #3521)
  – Remove unused data using a new `pio system prune` command (issue #3522)
  – Show ignored project environments only in the verbose mode (issue #3641)
  – Do not escape compiler arguments in VSCode template on Windows
  – Drop support for Python 2 and 3.5.

### 1.24.2 PlatformIO Core 4

See PlatformIO Core 4.0 history.

### 1.24.3 PlatformIO Core 3

See PlatformIO Core 3.0 history.

### 1.24.4 PlatformIO Core 2

See PlatformIO Core 2.0 history.

### 1.24.5 PlatformIO Core 1

See PlatformIO Core 1.0 history.

### 1.24.6 PlatformIO Core Preview

See PlatformIO Core Preview history.

### 1.25 Migrating from 4.x to 5.0

Guidance on how to upgrade from *PlatformIO Core (CLI)* v4.x to v5.x with emphasis on major changes, what is new, and what has been removed.

**PlatformIO Core 5.0 is fully backward compatible with PlatformIO 4.0 projects.**

Please read *PlatformIO 5.0 Release Notes* before.
1.25.1 Migration Steps


2. We recommend updating your project dependency declarations in `lib_deps` using a new owner-based syntax. See the `Library Manager` section for details.

1.25.2 What is new

In this section, we are going to highlight the most important changes and features introduced in PlatformIO Core (CLI) 5.0. Please visit `PlatformIO 5.0 Release Notes` for more detailed information.

PlatformIO Trusted Registry

PlatformIO Core 5.0 has been switched to the official PlatformIO Trusted Registry:

- Enterprise-grade package storage with high availability (multi replicas)
- Secure, fast, and reliable global content delivery network (CDN)
- Universal support for all packages:
  - Libraries
  - Development platforms
  - Toolchains
- Built-in fine-grained access control (role-based, `teams, organizations`).
The new Web front-end and upgraded PlatformIO Home are coming soon.

Collaborative Platform

PlatformIO Core 5.0 is fully unlocked for developers and teams. They can now share their packages (libraries, Development Platforms, toolchains) with team members or collaborate on open source projects. There are new CLI commands that help you to manage organizations, teams, team memberships, and resource access:

- `pio package` - manage packages in the registry
- `pio org` - manage organizations
- `pio team` - manage teams and team memberships
- `pio access` - manage package access for users, teams, and maintainers.

Package Management

The package management infrastructure has been rewritten from scratch. It is based now on the new PlatformIO Trusted Registry that supports a strict dependency declaration using the package owner. This improvement resolves the issues when package maintainers publish packages under the same name.

PlatformIO Core 5.0 does not handle packages from unofficial repositories declared via `packageRepositories` in `platform.json`. There were a lot of security issues and reports when PlatformIO Core 4.0 hangs when you manage external dependencies.

PlatformIO Core 5.0 uses THE ONLY official PlatformIO Trusted Registry that supports not only the libraries but also Development Platforms and toolchains.

Package maintainers can publish their libraries, development platforms, and toolchains to the registry using `pio package` CLI.

Library Manager

The biggest improvement for Library Management is the owner-based dependency declaration. You can finally forget about conflicts with library names in the registry. Use the new syntax `ownername/pkgname` to declare an owner-based dependency in "platformio.ini" (Project Configuration File) via `lib_deps`:

```
[env:myenv]
platform = ...
framework = ...
board = ...
lib_deps =
  bblanchon/ArduinoJson @ ^6.16.1
  knolleary/PubSubClient @ ^2.8
```

You can find an owner name of a library in the registry using PlatformIO Home > Libraries > Some Library > Installation tab.

Build System

SCons 4.0

PlatformIO Core 5.0 build engine has been upgraded to the latest SCons 4.0 - a next-generation software construction tool:
• *Configuration files are Python scripts* – use the power of a real programming language to solve build problems
• Built-in reliable and automatic dependency analysis
• Improved support for parallel builds
• Ability to *share built files in a cache* to speed up multiple builds.

**Custom Targets**

PlatformIO Core 5.0 gives more freedom to developers and *Development Platforms* maintainers. They can now declare the *Custom Targets:*

• Pre/Post processing based on dependent sources (another target, source file, etc.)
• Command launcher with own arguments
• Launch command with custom options declared in “platformio.ini” (*Project Configuration File*)
• Python callback as a target (use the power of Python interpreter and PlatformIO Build API)
• List available project targets (including dev-platform specific and custom targets) with a new `pio run --list-targets` command

See *Build System* section in *PlatformIO Core 5* release notes for more details.

### 1.25.3 What is changed or removed

**Drop support for Python 2 and 3.5**

Python 2.7 is reached the *end of its life* on 1 January 2020, and Python Software Foundation will not provide any security fixes for it. The same situation with Python 3.5.

To avoid unrelated issues to the PlatformIO Core, we decided to drop support for Python 2 and 3.5. *The minimum supported version for PlatformIO Core 5.0 is Python 3.6.*

If you use *PlatformIO IDE,* it already comes with the built-in compatible Python 3 interpreter. You do not need to do any extra steps. If you see a warning message that your local PlatformIO Core installation uses incompatible Python, please do the next steps:

1. Install the latest Python 3 following this guide *Install Python Interpreter*
2. Open system terminal and type `python3 --version` or `python.exe --version` (for Windows). The output should contain a version of Python 3.6 or above (depending on which you installed it).
3. Remove PlatformIO Core installation “penv” folder that is located by this path `USER_HOME_DIR/.platformio/penv`. If you use Windows and your user name contains non-ASCII chars the “penv” folder is located in `C:/platformio/penv`
4. Install PlatformIO Core using *Installer Script*
5. Run the `pio system info` command and ensure that Python 3 is used.

**Introducing Strict SSL/TLS**

The setting `strict_ssl` has been removed from *pio settings*. Now, PlatformIO Core 5.0 communicates over the encrypted SSL/TLS by default with the PlatformIO Registry and other services such as *Remote Development.*

At PlatformIO, we are always looking for ways to improve the security of our services.
**packageRepositories**

PlatformIO Core 5.0 does not support unofficial package repositories declared through `packageRepositories` in `platform.json` that was introduced in PlatformIO 3.0.

Please publish your development platforms and toolchains to the PlatformIO Trusted Registry using `pio package` CLI.

**Command Line Interface**

The following commands have been changed in v5.0.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pio access</code></td>
<td>New. Manage package access for users, teams, and maintainers</td>
</tr>
<tr>
<td><code>pio package</code></td>
<td>New. Manage packages in the registry (publish, unpublish)</td>
</tr>
<tr>
<td><code>pio project data</code></td>
<td>New. Dump build system data intended for IDE extensions/plugins</td>
</tr>
<tr>
<td><code>pio system info</code></td>
<td>New. Display system-wide information</td>
</tr>
<tr>
<td><code>pio system prune</code></td>
<td>New. Remove unused data</td>
</tr>
<tr>
<td><code>pio project init</code></td>
<td>Update project configuration for the specific environment using <code>pio project init --environment</code> option</td>
</tr>
<tr>
<td><code>pio run</code></td>
<td>List projects targets with <code>pio run --list-targets</code> option</td>
</tr>
<tr>
<td><code>pio account destroy</code></td>
<td>New command to remove permanently PlatformIO Account and related data</td>
</tr>
</tbody>
</table>
Symbols

--quiet
  pio-device-monitor command line
  option, 46
  pio-remote-device-monitor command
  line option, 113
-build-dir
  pio-ci command line option, 39
-core-packages
  pio-update command line option, 148
-current-password
  pio-account-register command line
  option, 33
-description
  pio-team-create command line
  option, 141
  pio-team-update command line
  option, 144
-dev
  pio-upgrade command line option, 152
-disable-auto-clean
  pio-remote-run command line option,
  116
  pio-run command line option, 125
-displayname
  pio-org-create command line option,
  81
  pio-org-update command line option,
  85
-dry-run
  pio-lib-update command line option,
  79
  pio-platform-update command line
  option, 103
  pio-remote-update command line
  option, 120
  pio-update command line option, 148
-dtr
  pio-device-monitor command line
  option, 45
  pio-remote-device-monitor command
  line option, 113
-echo
  pio-device-monitor command line
  option, 45
  pio-remote-device-monitor command
  line option, 113
-email
  pio-org-create command line option,
  81
  pio-org-update command line option,
  85
  -email, -e
    pio-account-register command line
    option, 31, 33
-encoding
  pio-device-monitor command line
  option, 46
  pio-remote-device-monitor command
  line option, 113
-env-prefix
  pio-project-init command line
  option, 107
-eol
  pio-device-monitor command line
  option, 46
  pio-remote-device-monitor command
  line option, 113
-exclude
  pio-ci command line option, 39
-exit-char
  pio-device-monitor command line
  option, 46
  pio-remote-device-monitor command
  line option, 113
-fail-on-defect
  pio-check command line option, 37
-firstname
  pio-account-register command line
  option, 46
  pio-remote-device-monitor command
  line option, 113
-flags
  pio-check command line option, 37
-force, -f
  pio-system-prune command line
  option, 139
-help, -h
  pio command line option, 21
-host
  pio-home command line option, 49
-id
  pio-lib-search command line option, 64
-ide
  pio-project-init command line
  option, 107
-installed
  pio-boards command line option, 34
-interface
  pio-debug command line option, 41
-json-output
  pio-access-list command line
  option, 23
  pio-account-show command line
  option, 32
  pio-account-token command line
  option, 32
  pio-boards command line option, 34
  pio-check command line option, 37
  pio-device-list command line
  option, 42
  pio-lib-builtin command line
  option, 51
  pio-lib-list command line option, 60
  pio-lib-search command line option, 64
  pio-lib-show command line option, 71, 73
  pio-lib-update command line option, 79
  pio-org-list command line option, 83
  pio-platform-frameworks command
  line option, 111
  pio-system-info command line
  option, 138
  pio-team-list command line option, 143
-keep-build-dir
  pio-ci command line option, 39
-lastname
  pio-account-register command line
  option, 31, 33
-list-targets
  pio-run command line option, 124
-logical
  pio-device-list command line
  option, 42
-mdns
  pio-device-list command line
  option, 42
-menu-char
  pio-device-monitor command line
  option, 46
  pio-remote-device-monitor command
  line option, 113
-monitor-dtr
  pio-test command line option, 147
-monitor-rts
  pio-test command line option, 147
-name
  pio-team-update command line
  option, 144
-no-ansi
  pio command line option, 21
-no-notify
  pio-package-publish command line
  option, 87
-no-open
  pio-home command line option, 49
-no-reset
  pio-test command line option, 146
-orgname
  pio-org-update command line option, 85
-owner
  pio-package-publish command line
  option, 87
-page
  pio-lib-search command line option, 64
-parity
  pio-device-monitor command line
  option, 45
-password, -p
pio-account-login command line option, 29
pio-account-register command line option, 31
-path
pio-system-completion-install command line option, 136
pio-system-completion-uninstall command line option, 137
-pattern
pio-check command line option, 37
-port
pio-home command line option, 49
-private
pio-package-publish command line option, 87
-raw
pio-device-monitor command line option, 46
pio-remote-device-monitor command line option, 113
-regenerate
pio-account-token command line option, 32
-released-at
pio-package-publish command line option, 87
-rts
pio-device-monitor command line option, 45
pio-remote-device-monitor command line option, 113
-rtscs
pio-device-monitor command line option, 45
pio-remote-device-monitor command line option, 113
-save / -no-save
pio-lib-install command line option, 56
pio-lib-uninstall command line option, 77
-serial
pio-device-list command line option, 42
-severity
pio-check command line option, 37
-shell
pio-system-completion-install command line option, 136
pio-system-completion-uninstall command line option, 137
-shutdown-timeout
pio-home command line option, 49
-skip-default
pio-platform-install command line option, 92
-storage
pio-lib-built-in command line option, 51
-test-port
pio-remote-test command line option, 119
pio-test command line option, 146
-type
pio-package-unpublish command line option, 88
-undo
pio-package-unpublish command line option, 88
-upload-port
pio-remote-run command line option, 116
pio-remote-test command line option, 119
pio-run command line option, 124
pio-test command line option, 146
-urn-type
pio-access-grant command line option, 22
pio-access-list command line option, 23
pio-access-private command line option, 25
pio-access-public command line option, 26
pio-access-revoke command line option, 27
-username, -u
pio-account-forgot command line option, 29
pio-account-login command line option, 29
pio-account-register command line option, 31, 33
-version
pio command line option, 21
-with-all-packages
pio-platform-install command line option, 92
-with-package
pio-platform-install command line option, 92
-without-building
pio-remote-test command line option, 119
pio-test command line option, 146
-without-package
pio-platform-install command line option, 92
-without-testing
   pio-test command line option, 146
-without-uploading
   pio-remote-test command line option, 119
   pio-test command line option, 146
-xonxoff
   pio-device-monitor command line option, 45
   pio-remote-device-monitor command line option, 113
-O, -project-option
   pio-ci command line option, 39
   pio-project-init command line option, 107
-a, -author
   pio-lib-search command line option, 64
-b, -baud
   pio-device-monitor command line option, 45
   pio-remote-device-monitor command line option, 113
-b, -board
   pio-ci command line option, 39
   pio-project-init command line option, 107
-c, -only-check
   pio-lib-update command line option, 79
   pio-platform-update command line option, 103
   pio-remote-update command line option, 120
   pio-update command line option, 148
-c, -project-conf
   pio-check command line option, 37
   pio-ci command line option, 39
   pio-debug command line option, 40
   pio-run command line option, 125
   pio-test command line option, 146
-d, -project-dir
   pio-check command line option, 37
   pio-debug command line option, 40
   pio-device-monitor command line option, 46
   pio-project-config command line option, 104
   pio-project-data command line option, 106
   pio-project-init command line option, 107
pio-remote-device-monitor command line option, 114
pio-remote-run command line option, 116
pio-remote-test command line option, 119
pio-run command line option, 125
pio-test command line option, 146
-d, -storage-dir
   pio-lib command line option, 50
-d, -working-dir
   pio-remote-agent-start command line option, 110
-e, -environment
   pio-check command line option, 37
   pio-debug command line option, 40
   pio-device-monitor command line option, 46
   pio-lib command line option, 50
   pio-project-data command line option, 106
   pio-project-init command line option, 107
   pio-remote-device-monitor command line option, 113
   pio-remote-run command line option, 114
   pio-remote-test command line option, 118
   pio-run command line option, 124
   pio-test command line option, 145
-f, -filter
   pio-device-monitor command line option, 46
   pio-remote-device-monitor command line option, 113
   pio-test command line option, 145
-f, -force
   pio-lib-install command line option, 56
   pio-lib-uninstall command line option, 77
   pio-platform-install command line option, 92
-f, -framework
   pio-lib-search command line option, 64
-g, -global
   pio-lib command line option, 50
-i, -header
   pio-lib-search command line option, 64
-i, -ignore
   pio-remote-test command line
environment variable
CI, 293
PLATFORMIO_AUTH_TOKEN, 29, 32, 294, 2606, 2741
PLATFORMIO_BOARDS_DIR, 263, 295
PLATFORMIO_BUILD_CACHE_DIR, 260, 294
PLATFORMIO_BUILD_DIR, 261, 295
PLATFORMIO_BUILD_FLAGS, 269, 295, 2727, 2735, 2743
PLATFORMIO_CACHE_DIR, 260, 294
PLATFORMIO_CORE_DIR, 259, 294, 2760
PLATFORMIO_DATA_DIR, 262, 295
PLATFORMIO_DEFAULT_ENVS, 257, 296
PLATFORMIO_DISABLE_PROGRESSBAR, 294
PLATFORMIO_EXTRA_SCRIPTS, 296, 298
PLATFORMIO_FORCE_ANSI, 21, 294
PLATFORMIO_GLOBALLIB_DIR, 259, 294
PLATFORMIO_INCLUDE_DIR, 261, 294
PLATFORMIO_LIB_DIR, 262, 295
PLATFORMIO_LIB_EXTRA_DIRS, 274, 295
PLATFORMIO_LIBDEPS_DIR, 261, 295
PLATFORMIO_NO_ANSI, 21, 294
PLATFORMIO_PACKAGES_DIR, 259, 294
PLATFORMIO_PLATFORMS_DIR, 259, 294
PLATFORMIO_REMOTE_AGENT_DIR, 295
PLATFORMIO_SETTING_AUTO_UPDATE_LIBRARIES, 296
PLATFORMIO_SETTING_AUTO_UPDATE_PLATFORMS, 296
PLATFORMIO_SETTING_CHECK_LIBRARIES_INTERVAL, 296
PLATFORMIO_SETTING_CHECK_PLATFORMIO_INTERVAL, 296
PLATFORMIO_SETTING_CHECK_PLATFORMS_INTERVAL, 296
PLATFORMIO_SETTING_DISABLE_CACHE, 296
PLATFORMIO_SETTING_DISABLE_TELEMETRY, 296
PLATFORMIO_SETTING_FORCE_VERBOSE, 38, 40, 41, 116, 119, 125, 147, 296
PLATFORMIO_SETTING_PROJECTS_DIR, 296
PLATFORMIO_SETTING_PROJECTS_DIR, 296
PLATFORMIO_SHARED_DIR, 263, 295
PLATFORMIO_SRC_BUILD_FLAGS, 271, 295
PLATFORMIO_SRC_DIR, 261, 294
PLATFORMIO_SRC_FILTER, 272, 295
PLATFORMIO_TEST_DIR, 262, 295
PLATFORMIO_UPLOAD_FLAGS, 277, 296
pio command line option
-`help`, `-h`, 21
-`no-ansi`, 21
-`version`, 21
pio-access-grant command line option
-`urn-type`, 22
pio-access-list command line option
-`json-output`, 23
-`urn-type`, 23
pio-access-private command line option
-`urn-type`, 25
pio-access-public command line option
-`urn-type`, 26
pio-access-revoke command line option
-`urn-type`, 27
pio-account-forgot command line option
-`username`, `-u`, 29
pio-account-login command line option
-`password`, `-p`, 29
-`username`, `-u`, 29
pio-account-register command line option
-`current-password`, 33
-`email`, `-e`, 31, 33
-`firstname`, 31, 33
-`lastname`, 31, 33
-`password`, `-p`, 31
-`username`, `-u`, 31, 33
pio-account-show command line option
-`json-output`, 32
pio-account-token command line option
-`json-output`, 32
-`regenerate`, 32
pio-boards command line option
-`installed`, 34
-`json-output`, 34
pio-check command line option
-`fail-on-defect`, 37
-`flags`, 37
-`json-output`, 37
-`pattern`, 37
-`severity`, 37
-`c`, `-project-conf`, 37
-`d`, `-project-dir`, 37
-`e`, `-environment`, 37
-`s`, `-silent`, 38
-`v`, `-verbose`, 38
pio-ci command line option
-`build-dir`, 39
-`exclude`, 39
-`keep-build-dir`, 39
-`O`, `-project-option`, 39
-`b`, `-board`, 39
-`c`, `-project-conf`, 39
-`l`, `-lib`, 39
-`v`, `-verbose`, 40
pio-debug command line option
-`interface`, 41
-`c`, `-project-conf`, 40
-`d`, `-project-dir`, 40
-`e`, `-environment`, 40
-`v`, `-verbose`, 41
pio-device-list command line option
-`json-output`, 42
-`logical`, 42
-`mdns`, 42
-`serial`, 42
pio-device-monitor command line option
--`quiet`, 46
-`dtr`, 45
-`echo`, 45
-`encoding`, 46
-`eol`, 46
-`exit-char`, 46
-`menu-char`, 46
-`parity`, 45
-`raw`, 46
-`rts`, 45
-`rtscts`, 45
-`xonxoff`, 45
-`b`, `-baud`, 45
-`d`, `-project-dir`, 46
-`e`, `-environment`, 46
-`f`, `-filter`, 46
-`p`, `-port`, 45
pio-home command line option
-`host`, 49
-`no-open`, 49
-`port`, 49
-`shutdown-timeout`, 49
pio-lib command line option
-`d`, `-storage-dir`, 50
-`e`, `-environment`, 50
-`g`, `-global`, 50
pio-lib-builtin command line option
-`storage`, 51
pio-lib-install command line option
-`save / -no-save`, 56
-`f`, `-force`, 56
-`s`, `-silent`, 56
pio-lib-list command line option
-`json-output`, 60
pio-lib-search command line option
-id, 64
-json-output, 64
-page, 64
-a, -author, 64
-f, -framework, 64
-i, -header, 64
-k, -keyword, 64
-n, -name, 64
-p, -platform, 64
pio-lib-show command line option
-json-output, 71, 73
pio-lib-uninstall command line option
-save / -no-save, 77
-f, -force, 77
-s, -silent, 77
pio-lib-update command line option
-dry-run, 79
-json-output, 79
-c, -only-check, 79
pio-org-create command line option
-displayname, 81
-email, 81
pio-org-list command line option
-json-output, 83
pio-org-update command line option
-displayname, 85
-email, 85
-orgname, 85
pio-package-pack command line option
-o, -output, 86
pio-package-publish command line option
--no-notify, 87
-owner, 87
-private, 87
-released-at, 87
pio-package-unpublish command line option
-type, 88
-undo, 88
pio-platform-frameworks command line option
-json-output, 89
pio-platform-install command line option
-skip-default, 92
-with-all-packages, 92
-with-package, 92
-without-package, 92
-f, -force, 92
pio-platform-list command line option
-json-output, 96
pio-platform-search command line option
-json-output, 97
pio-platform-update command line option
-dry-run, 103
-json-output, 103
-c, -only-check, 103
-p, -only-packages, 102
pio-project-config command line option
-json-output, 104
-d, -project-dir, 104
pio-project-data command line option
-json-output, 106
-d, -project-dir, 106
-e, -environment, 106
pio-project-init command line option
-env-prefix, 107
-ide, 107
-o, -project-option, 107
-b, -board, 107
-d, -project-dir, 107
-e, -environment, 107
-s, -silent, 107
pio-remote-agent-start command line option
-d, -working-dir, 110
-n, -name, 110
-s, -share, 110
pio-remote-device-list command line option
-json-output, 111
pio-remote-device-monitor command line option
--quiet, 113
dtr, 113
echo, 113
encoding, 113
eol, 113
exit-char, 113
menu-char, 113
parity, 113
raw, 113
rts, 113
rtscts, 113
xonxoff, 113
-b, -baud, 113
d, -project-dir, 114
-e, -environment, 114
-f, -filter, 113
-p, -port, 113
pio-remote-run command line option
--disable-auto-clean, 116
upload-port, 116
d, -project-dir, 116
-e, -environment, 116
-r, -force-remote, 116
-t, -target, 116
-v, -verbose, 116
pio-remote-test command line option
-test-port, 119
-upload-port, 119
-without-building, 119
-without-uploading, 119
-d, -project-dir, 119
-e, -environment, 118
-i, -ignore, 119
-r, -force-remote, 119
-v, -verbose, 119
pio-remote-update command line option
-dry-run, 120
-c, -only-check, 120
pio-run command line option
-disable-auto-clean, 125
-list-targets, 124
-upload-port, 124
-c, -project-conf, 125
-d, -project-dir, 125
-e, -environment, 124
-j, -jobs, 125
-s, -silent, 125
-t, -target, 124
-v, -verbose, 125
pio-system-completion-install command line option
-path, 136
-shell, 136
pio-system-completion-uninstall command line option
-path, 137
-shell, 137
pio-system-info command line option
-json-output, 138
pio-system-prune command line option
-force, -f, 139
pio-team-create command line option
-description, 141
pio-team-list command line option
-json-output, 143
pio-team-update command line option
-description, 144
-name, 144
pio-test command line option
-monitor-dtr, 147
-monitor-rts, 147
-no-reset, 146
-test-port, 146
-upload-port, 146
-without-building, 146
-without-uploading, 146
-c, -project-conf, 146
-d, -project-dir, 146
-e, -environment, 145
-f, -filter, 145
-i, -ignore, 146
-v, -verbose, 147
pio-update command line option
-core-packages, 148
-dry-run, 148
-c, -only-check, 148
pio-upgrade command line option
-dev, 152
PLATFORMIO_AUTH_TOKEN, 29, 32, 2606, 2741
PLATFORMIO_BOARDS_DIR, 263
PLATFORMIO_BUILD_CACHE_DIR, 260
PLATFORMIO_BUILD_DIR, 261
PLATFORMIO_BUILD_FLAGS, 269, 2727, 2735, 2743
PLATFORMIO_CACHE_DIR, 260
PLATFORMIO_CORE_DIR, 259, 2760
PLATFORMIO_DATA_DIR, 262
PLATFORMIO_DEFAULT_ENVS, 257
PLATFORMIO_DISABLE_PROGRESSBAR, 294
PLATFORMIO_EXTRA_SCRIPTS, 298
PLATFORMIO_FORCE_ANSI, 21
PLATFORMIO_GLOBALLIB_DIR, 259
PLATFORMIO_INCLUDE_DIR, 261
PLATFORMIO_LIB_DIR, 262
PLATFORMIO_LIB_EXTRA_DIRS, 274
PLATFORMIO_LIBDEPS_DIR, 261
PLATFORMIO_NO_ANSI, 21
PLATFORMIO_PACKAGES_DIR, 259
PLATFORMIO_PLATFORMS_DIR, 259
PLATFORMIO_SETTING_FORCE_VERBOSE, 38, 40, 41, 116, 119, 125, 147
PLATFORMIO_SHARED_DIR, 263
PLATFORMIO_SRC_BUILD_FLAGS, 271
PLATFORMIO_SRC_DIR, 261
PLATFORMIO_SRC_FILTER, 272
PLATFORMIO_TEST_DIR, 262
PLATFORMIO_UPGRADE_FLAGS, 277
PLATFORMIO_UPLOAD_PORT, 276
PLATFORMIO_WORKSPACE_DIR, 256, 260